НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Система моніторингу використання зображень у соціальній мережі Reddit»

Виконав:
студент IV курсу, групи КП-52
Шароварський Костянтин Вікторович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Сулема Є.С. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
в. о. завідувача кафедри ММСА ІПСА, к.т.н,
Тимощук О.Л. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Шароварському Костянтину Вікторовичу

1. Тема проекту «Система моніторингу використання зображень соціальної мережі Reddit», керівник проекту Сулема Євгенія Станіславівна, к.т.н., доцент, затверджені наказом по університету від «22» травня 2019 р. №1331-С

2. Термін подання студентом проекту «19» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень;
   - розроблення системи моніторингу;
   - опис розроблених алгоритмів та підпрограм;
   - аналіз розробленої системи.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма класів алгоритму виділення метаданих зображення (креслення);
   - діаграма розгортання системи моніторингу (креслення);
   - результат оптимізації функції обробки зображень (плакат);
   - процес перетворення зображення в гістограми та порівняння (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	14.11.2018	
2.	Розроблення та узгодження технічного завдання	28.11.2018	
3.	Розроблення структури системи моніторингу	15.12.2018	
4.	Підготовка матеріалів першого розділу дипломного проекту	30.12.2018	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2019	
6.	Підготовка матеріалів другого розділу дипломного проекту	20.02.2019	
7.	Програмна реалізація системи моніторингу	10.03.2019	
8.	Тестування системи моніторингу	17.03.2019	
9.	Підготовка матеріалів третього розділу дипломного проекту	30.03.2019	
10.	Підготовка матеріалів четвертого розділу дипломного проекту	11.04.2019	
11.	Підготовка графічної частини дипломного проекту	21.04.2019	
12.	Оформлення документації дипломного проекту	26.05.2019	

Студент К.В. Шароварський

Керівник проекту Є.С. Сулема

АНОТАЦІЯ

Дана дипломна робота присвячена розробленню системи, що проводить моніторинг використання зображень в соціальній мережі Reddit та надає інструменти для подальшого аналізу цієї інформації.

В роботі виконано аналіз існуючих аналогів та алгоритмів пошуку схожих зображень. Побудована архітектура сприяє масштабуванню системи при різних навантаженнях та кількості зображень, що публікуються в соціальній мережі. Був реалізований алгоритм виділення метаданих та протестований на предмет швидкодії та використання пам’яті. На основі цих метаданих працює алгоритм пошуку, який було розроблено на основі особливих технологій бази даних MongoDB. Ці функціональні можливості доступні за допомогою web-інтерфейсу. Всі можливості системи доступні тільки авторизованому користувачу.

Система використовує Reddit API для завантаження нових публікацій та ставить в чергу знайдені зображення для подальшої обробки. Оброблені зображення є доступними кінцевому користувачу. За допомогою системи, користувач має можливість шукати та переглядати статистику вже нових даних.

ABSTRACT

This diploma project is dedicated to developing the system which performs monitoring of image usage in Reddit social network and gives the tools to analyze this information further.

In scope of the project alternative solutions and similar image search algorithms were analyzed. Developed architecture of the system gives it an ability to scale under different load and amount of image data which is published in the social network. Algorithm of image metadata extraction was developed and tested regarding speed and memory usage. Extracted metadata was used in search algorithm which was developed based on special features of MongoDB database. This functionality is presented to the user as a web interface. All features of the application are protected from unauthorized access.

System utilizes Reddit API to fetch new publications and queue up any images found to be processed. Processed images are then immediately available to the end user. The system provides that user with the functionality to search and see statistics of the new data.

ДП.045440-01-90 Система моніторингу використання зображень у соціальній мережі Reddit. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Технічне завдання	4	
ДП.045440-03-81	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Пояснювальна		
	записка	61	
ДП.045440-04-51	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Програма та		
	методика тестування	4	
ДП.045440-05-34	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Керівництво		
	користувача	8	
ДП.045440-06-99	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Структура класів		
	алгоритму виділення		
	метаданих. Діаграма		
	класів	1	

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Розгортання		
	системи моніторингу.		
	Схема розгортання	1	
ДП.045440-08-98	Система моніторингу		
	використання зображень		
	у соціальній мережі		
	Reddit. Компакт-диск	1	

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

СИСТЕМА МОНІТОРИНГУ ВИКОРИСТАННЯ ЗОБРАЖЕНЬ У СОЦІАЛЬНІЙ МЕРЕЖІ Reddit

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____К.В Шароварський

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Система моніторингу використання зображень у соціальній мережі Reddit.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання користувачами соціальної мережі Reddit. Такі користувачі матимуть можливість проаналізувати опубліковані іншими користувачами зображення з метою отримання додаткової інформації про мережу.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Система моніторингу повинна забезпечувати наступні основні функції:

1) Можливість автентифікації та реєстрації
2) Підтвердження реєстрації за допомогою електронної пошти
3) Авторизація користувачів при кожному доступі до даних
4) Збір публікацій із зображеннями з соціальної мережі Reddit у реальному часі
5) Завантаження зображення та пошук схожих до нього
6) Перегляд статистики збору зображень
7) Перегляд найчастіше опублікованих зображень
8) Фільтрація пошуку та статистики по тематичним підсайтам соціальної мережі Reddit.

Також, були поставлені такі додаткові вимоги:

1) Кросплатформеність
2) Масштабованість

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структура класів алгоритму виділення метаданих зображення. Діаграма класів»;
   - «Розгортання системи моніторингу. Схема розгортання».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2018

Розроблення та узгодження технічного завдання……....28.11.2018

Розроблення структури системи моніторингу……....15.12.2018

Підготовка матеріалів першого розділу дипломного проекту…….03.02.2019

Програмна реалізація системи…..……....17.03.2019

Тестування системи……..……..…….....03.04.2019

Підготовка матеріалів текстової частини проекту……..….28.04.2019

Підготовка матеріалів графічної частини проекту……..…12.05.2019

Оформлення технічної документації проекту……..…25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

СИСТЕМА МОНІТОРИНГУ ВИКОРИСТАННЯ ЗОБРАЖЕНЬ У СОЦІАЛЬНІЙ МЕРЕЖІ Reddit

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ К.В. Шароварський

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Сабредіт – тематичний розділ соціальної мережі Reddit зі своїми публікаціями та модераторами

Модератор – людина, що слідкує за виконанням правил поведінки в сабердіті

Beta – бета тестування, стадія розробки програмного забезпечення, на якій застосунок ще має велику кількість відомих та невідомих дефектів.

Індексація – процес, під час якого автоматичний процес збирає інформацію про ресурс, для подальшого використання її під час пошуку.

Big data – технології значного прискорення обробки великих об’ємів інформації

Python – мова програмування, що інтерпретується та має динамічну типізацію

.NET Framework – фреймворк від компанії Microsoft, який можна використовувати у мовах програмування C# та F#

HTML – HyperText Markup Language, мова розмітки веб сторінки

JavaScript – мова програмування у веб браузері

ASP.NET – фреймворк віл компанії Microsoft розробки веб застосунків

ASP.NET Core – наступна ітерація розробки ASP.NET із відкритим кодом та великою швидкодією від компанії Microsoft

Spring – фреймворк веб розробки для розробки на мові Java

Kestrel – веб сервер, розроблений Microsoft, із направленням на швидкодію

AWS – Amazon Web Services, компанія, яка надає послуги провайдера хмарних технологій

CDN – Content Delivery Network, розподілена сітка дата центрів та проксі серверів, які спрямовані на швидку доставку об’єктів (картинок, текстів, скриптів) користувачам

S3 – Simple Storage Service, розподілена файлове сховище

SQS – Simple Queue Service, розподілена черга від компанії Amazon

EC2 – Elastic Compute Cloud, розподілений сервіс хмарних обчислень

SSH – Secure Shell, протокол для віддаленої комунікації із пристроями

JSON – JavaScript Object Notation, формат збереження та передачі даних

NPM – Node Package Manager, система для збереження, поширення та доступу до пакунків мови JavaScript

RGB – кольорова модель, яка розглядає колір, як складові червоної, синьої та зеленої складових

NSFW – Not Safe for Work, посилання, які небезпечно переглядати

JWT – JSON Web Token, стандарт для безпечних токенів на базу JSON

HTTP – HyperText Transfer Protocol, протокол передачі даних для гіпертекстових даних

MFA – Multi-factor authentication, автентифікація користувачів, що складається з декількох факторів

КБ – кілобайт, що дорівнює 1024 байтам

ГБ – гігабайт, що дорівнює 1024 кілобайтам

ВСТУП

Сучасне суспільство все більше і більше опирається на соціальні мережі як засіб комунікації людей. Використовуються повідомлення та публікації на відміну від передачі такої інформації за допомогою слів при особистій зустрічі. Це обумовлено тим, що за допомогою інтернет технологій аудиторія, яка отримає інформацію значно розширюється, порівнюючи з кількістю людей, які зможуть прослухати інформацію знаходячись поряд із доповідачем.

Але таким чином передають не тільки текстову інформацію, а ще й мультимедійну, таку як зображення. Через масштабність соціальних мереж, мультимедійна інформація досягає величезних розмірів та проаналізувати її будь-яким чином без певних технічних застосунків за допомогою малої кількості людей майже неможливо.

Такий технічний застосунок повинен мати можливості збору мультимедійних даних та інструментів для роботи над ними. Він дозволить зацікавленим людям оброблювати величезні об’єми мультимедійної інформації, які неможливо продивитись людині без технічної підтримки.

Отже, створення застосунку, який збирає інформацію в певній соціальній мережі про зображення є актуальною задачею.

Даний дипломний проект присвячено розробленню системи моніторингу зображень в соціальній мережі Reddit. Така система призначення для користувачів цієї соціальної мережі для отримання докладнішої інформації про опубліковані зображення.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1. Загальний опис проблеми пошуку в соціальній мережі Reddit

Соціальні мережі дуже відомі своєю популярністю. В таких мережах беруть участь величезна кількість людей. Наприклад, соціальною мережею Facebook на даний момент користуються якнайменше 2 міліарди людей. Навіть таку нишеву мережу як Reddit кожного місяця відвідують понад 330 мільйонів унікальних користувачів. Кожну годину, в середньому, створюється понад 2000 публікацій [1], не враховуючи коментарів. Через таку приголомшливу кількість інформації, слідкувати за усім неможливо. Єдиний спосіб мати хоч якусь можливість знаходити щось потрібне – за допомогою пошуку. Отже, пошук – дуже важлива частина популярної соціальної мережі.

Через важливість функції пошуку, його якість повинна бути відповідною. Користувач повинен мати змогу шукати найзручнішим для нього способом. Результати повинні бути якнайбільше релевантні запиту людини. Задача досягання таких результатів надзвичайно складна, вирішити проблему пошуку намагається велика кількість компаній: Google, Bing, Baidu, Yandex та ін.

У соціальній мережі Reddit, усупереч попиту, система пошуку не є доскональною. Текстовий пошук не дає можливості обрати із запропонованих ключові слова. Також, основним критерієм пошуку є нещодавність публікації та її оцінка. Частота входу пошукового запиту до тексту публікації ніяк не впливає на результат.

Але даний проект розгляне детальніше іншу проблему – проблему пошуку зображень. На даний момент, ця соціальна мережа не передбачає ніяких можливостей мультимедійного пошуку. Тобто, знайти якусь публікацію, якщо користувач знає як точно виглядає зображення, неможливо.

Виділимо дві підпроблеми:

1) Відсутність функції пошуку схожих зображень.

   Існує декілька прикладів використання такої функції системи. По-перше, завжди існує проблема плагіату. В системах з повною свободою публікації вона ще більше впливова через велику кількість інформації. Можливість задати певне зображення як фільтр для пошуку надасть користувачам змогу знайти людей, що недоброякісно користуються чужим контентом. По-друге, така функція спростить пошук релевантної інформації. Наприклад, якщо людина виявить бажання знайти інформацію по певній будівлі, тоді така фільтрація сильно допоможе.

2) Неможливість виділення груп зображень.

   Так як людина не може сприймати тисячі зображень одночасно, то інформацію по загальному використанню зображень потрібно якось групувати. В цьому випадку, переглянувши найчисленніші або найважніші групи зображень, цей користувач зможе зробити певні висновки щодо трендів у певному підрозділі соціальної мережі.

Через складність всієї проблеми, у даній роботі було вирішено сконцентруватися на частині, що зв’язані з зображеннями. Проблеми, зв’язані з текстовим пошуком, є за межею даного дипломного проекту.

1.2. Аналіз існуючих аналогів

Аналогів для рішення описаної проблеми існує небагато. Розглянемо декілька з них.

1.2.1. *Karmadecay*

Веб сайт Karmadecay [2] нещодавно з’явився в Інтернеті та знаходиться на стадії beta. Він включає в себе наступні можливості:

1) Фільтрація.

Застосунок підтримує декілька опцій для фільтрації. Першою є опція використовувати NSFW зображення в результатах. Другою опцією є список сабредитів, в яких виконувати пошук. Можна тільки чотири задані сабредіти, або всю соціальну мережу

2) Завантаження файла та пошук.

Вище зазначені фільтри застосуються до пошуку по зображенню, що повинне бути завантажене. Також, можна ввести посилання на зображення.

3) Перегляд останніх 5 дублікатів.

Також, сторінка зображує останні 5 зображень, що були публіковані декілька разів у соціальній мережі.

Одним із недоліків даного сервісу є алгоритм перевірки схожості. Експериментально було перевірено, що Karmadecay перевіряє на майже повну співпадіння зображень. Тобто, якщо зображення схожі менше ніж приблизно 98%, то сервіс буде відображати пустий результат пошуку.

Це було перевірено завантаживши зображення та провівши операцію пошуку по зображенням, на яке було додано одну тонку додаткову чорну лінію. В результаті, після ініціації процесу пошуку, жодного результату не було знайдено.

Другим недоліком є значно обмежені функціональні можливості фільтрації. На вибір тематичних сторінок надано тільки 4 опції. У випадку, якщо автор забажає створити нову сторінку, тоді вона буде відсутньою в запропонованих опціях фільтрації. Це сильно звужує можливості для користувачів, що бажають виконувати операції пошуку та інші можливості системи тільки в контексті певної тематичної частини соціальної мережі, якою ці користувачі зацікавлені.

По третє, сервіс збирає інформацію тільки з популярних частин web-сайту. Сервіс не виконує індексацію зображень, сабредіт яких не є популярним.

Також, період індексації зображень є доволі великим. З моменту публікації повинен пройти якнайменше 1 година перед тим, як результати будуть доступні на сервісі.

1.2.2. *Google Images*

Компанія Google має безліч сервісів, які вона надає користувачам. Google Images – один із них, що дає можливість пошуку по завантаженому зображенню.

Компанія надає зручний та широкий механізм пошуку. За допомогою нього, можна імітувати пошук схожих зображень на Reddit. Для цього, після завантаження фото для пошуку, потрібно додати *site:reddit.com* до запиту. Після цих маніпуляцій, Google Images [3] почне шукати схожі картинки на цьому веб сайті.

Однією з переваг цього сервісу є дуже гнучкі варіанти налаштувань. На вибір користувачу надаються наступні інструмента налаштування:

1) Розмір.
2) Колір.
3) Права на використання.
4) Тип зображення.
5) Час публікації.

Також, Google використовує свої потужні алгоритми та обладнання, які тестувались на найголовнішому для компанії сервісі пошуку тексту, що робить пошук зображень також надзвичайно швидким. При незадовільних результатах пошуку можна додати уточнюючі ключові слова.

Отже, можна виділити наступні переваги:

1) Дуже швидкий пошук.
2) Пошук знаходить навість частково схожі зображення.
3) Можна додати певні ключові слова до запиту для більш точного знаходження результату.
4) Гнучкі налаштування.

Недоліком цієї системи є те, що вона не сконцентрована на певному web-сайті, а призначена для всієї мережі Інтернет. Тобто, Google Images не надає контексну інформацію про зображення, наприклад автора публікації, назву тематичного підсайту та ін. Також через це система надзвичайно навантажена через необхідність індексувати всі web-сайти, тому інформація з конкретного web-сайту Reddit надходитиме з певною затримкою.

Підсумуючи, виділемо такі недоліки:

1) Немає можливості перегляду останніх трендів використання зображень в публікаціях.
2) Індексація нових зображень проходить дуже довго через величезне навантаження на сервери Google [4].
3) Не можна переглянути усі публікації із зображенням.

Отже, спеціалізований веб сайт Karmadecay ще не є достатньо розвинутим для використання. А інструмент широкого призначення Google Images не має спеціальних функцій, що б вирішили проблеми пошуку саме у соціальній мережі Reddit.

1.3. Аналіз існуючих алгоритмів пошуку зображень

Науковці зі всього світу займаються різними дослідженнями по темі аналізу зображень. Розглянемо декілька розроблених підходів щодо оцінки схожості зображень.

Наприклад, одним з таких алгоритмів є OASIS – онлайн алгоритм для масштабованої перевірки схожості зображень [5]. Автори стверджують, що за 3 дні цей алгоритм можна натренувати на 1.2 мільйонах зображень. Він заснований на пасивно агресивних алгоритмах навчання. Повне зображення поділяють на квадратні області. Далі, для репрезентації кожної такої області використовується комбінація крайової карти та карти кольорів.

Інші алгоритми засновані на роботі штучних нейронних мереж, які зараз є досить популярними. Цим користується алгоритм від Нортвестернського університету, компанії Google та Каліфорнійського

інституту технологій [6]. В реалізації він використовує багато масштабну архітектуру нейронної мережі з великою кількістю шарів. Кожна картинка ставиться у відповідність позитивній картинці, тобто такій, що схожа до даної. Також, картинка ставиться у відповідність негативній картинці, що не схожа до даної. Такий елемент називається трійкою та за допомогою нього можна навчити мережу сприймати і позитивні, і негативні результати.

Схожість зображення іноді потрібно обчислювати в різних спектрах. Вчені з'ясували, що нейронні мережі, натреновані на даних з одного спектру, можуть знаходити схожі елементи в зображеннях певного іншого спектру [7].

Недоліками нейронних мереж є те, що їх складно навчати на зображеннях у реальному часі, а у випадку з моніторингом соціальної мережі, це є основною задачею.

Також, є інші, алгоритмічні підходи до оцінювання наскільки два зображення є схожими.

Гістограми кольорів пропонуються дослідниками з університету Амраваті в Індії [8]. Вони пропонують виділяти з кожного зображення гістограму кольорів, що є менша за розміром, ніж повне зображення та достатньо характеризує розподіл кольорів по пікселях на зображенні. Далі, за допомогою функції порівняння гістограм обчислити значення функції відстані між цими гістограмами. Далі, зображення із найменшими відстанями і є шуканими схожими зображеннями.

Такий підхід має алгоритмічну складність $O(n)$, тобто лінійну. Він має таку складність, адже для побудови гістограм потрібно прочитати всі пікселі зображення лише один раз.

Отже, враховуючи швидкодію останнього алгоритму, оберемо метод гістограм для пошуку схожих зображень.

1.4. Обґрунтування актуальності теми

За останні роки кількість інформації збільшується неймовірними темпами. Вона генерується і людьми, і машинами. Розв'язки задач обробки великої кількості даних є необхідними для прискорення засвоєння нового.

Як було зазначено, соціальна мережа Reddit не є винятком у тренді завантаженості контентом. Для вирішення проблем, поставлених вище, необхідно аналізувати всі створюємі публікації у режимі реального часу. Для цього потрібно розробити алгоритми та підходи для маніпулювання великою кількістю даних.

Алгоритми обробки великої кількості даних, або big data, мають велику популярність у сучасному світі. Багато вчених та інженерів вирішують певні підзадачі цього сегменту кожного дня.

Обробка зображень у реальному часі – ще складніша задача. Для розв’язку цієї частини проблеми, потрібно використати ті знання, що притаманні конкретно зображенням.

Також можна зазначити, що даний проект дозволить протестувати алгоритми обробки зображень на основі практичної задачі. Доволі часто, дослідники перевіряють свої алгоритми на перевірочному наборі даних, який був підібраний ними. У випадку з моніторингом контенту соціальної мережі, користувачі надають зображення для аналізу. Тобто, дані, що будуть використовуватися в даному проекті є прямим відображенням того, чим повсякденно користуються люди.

Отже, розроблення алгоритмів обробки великої кількості зображень у реальному часі є актуальною темою та дає користувачам необхідні інструменти для аналізу значного об’єму даних.

1.5. Висновки

В даному розділі було виділено такі проблеми соціальної мережі Reddit: відсутність функціональної можливості переглянути схожі зображення та неможливість переглянути найчисленніші групи зображень.

Були визначені аналоги, які включають в себе сервіс Karmadecay та Google Images. Ці програмні продукти було проаналізовано та визначено їх переваги та недоліки.

Проаналізувавши існуючі алгоритми пошуку, було визначено, що за специфікою даного проекту найкраще підходить метод гістограм. Ключовою характеристикою при виборі була швидкодія обробки зображення. Швидкодія є надзвичайно важливою характеристикою через величезну кількість даних, що поступають в реальному часі від користувачів.

2. РОЗРОБЛЕННЯ СИСТЕМИ МОНІТОРИНГУ

2.1. Аналіз вимог до функціональності системи

Поділимо весь аналіз вимог на дві частини: аналіз вимог від зацікавлених сторін та технічні вимоги застосунку.

2.1.1. *Зацікавлені сторони*

Виділимо зацікавлені сторони проекту.

1) Автор публікацій у соціальній мережі Reddit.

   Автор публікацій – користувач, що публікує власні твори у соціальній мережі Reddit. Проблема автору в тому, що публікації у вигляді картинки дуже просто скопіювати іншим користувачам та видати за свій твір. Прикладом таких творів може бути: фотографія природи, фотографія фізичної картини, картина створена мультимедійними програмами тощо.

   Отже, від продукту, що збирає інформацію про використання зображень, автор публікацій вимагатиме можливість знайти публікації, що дуже нагадують авторську заради боротьби з недобросовісними користувачами.

2) Користувач, що проглядає публікації.

   Даний тип користувача не створює власні публікації, а лише споживає твори інших заради розваг чи пошуку знань. Іноді такий тип користувачів називають «спостергігачем». «Спостерігач» стикається з проблемою пошуку інформації, так як мережа Reddit надає можливість шукати тільки текстову інформацію.

   Вимогою такого користувача є можливість завантажити картинку, шукати схожі до неї.

3) Модератор тематичного підсайту Reddit, або сабредіта.

   Задача модераторів є моніторинг виділеної ділянки веб-сайта. Їх прямий обов’язок – це контролювати користувачів, щоб вони не

порушували правила сабредіту. Вічною проблемою відкритих платформ є «спам» – масове розсилання публікацій, які не мають цінності, а лише засмічують інформаційний простір такої платформи. Модератори сабредіту вимагають можливість відслудкувати кластери ідентичного контенту за певний час заради кращого розуміння ситуації у сабредіті.

Отже, виділимо вимоги високого рівня, або бізнес-вимоги:

1) Пошук схожих публікацій.
2) Виділення кластерів схожого контенту.

2.1.2. *Технічні вимоги*

Даний застосунок буде працювати з великою кількістю зображень, та він має підтримувати обробку потоку даних в реальному часі. Тому, архітектура процесів повинна підтримувати горизонтальне масштабування. Наприклад, якщо у 2019 році у середньому в соціальну мережу надсилають близько 3 картинок в секунду, то якщо в 2020 році потік стане більше в два рази, то система повинна почати підтримувати збільшений потік даних за допомогою підключення додаткових машин з обчислювальними ресурсами.

Також, зараз можна вважати стандартною практику вимагання реєстрації та автентифікації користувачів для розмежування прав та можливості відключити недобросовісних користувачів. Виходячи з цього, система моніторингу повинна також містити підсистеми авторизації користувачів.

Кросплатформеність значно розширяє можливості розташування застосунку на зовнішніх сервісах. Ця властивість зменшує ціну хостингу та дає можливість використати сервіси, що базуються на різних операційних системах. Однією з вимог до застосунку є кросплатформеність.

Підсумуючи, до застосунку задаються наступні технічні вимоги:

1) Горизонтальна масштабованість процесу обробки зображень.
2) Реєстрація та авторизація користувачів.

3) Кросплатформеність усіх підпрограм.

2.2. Обґрунтування вибору технологій розробки системи

Поділимо технології розробки системи на клієнтську частину, хмарного провайдера та серверну частину

2.2.1. *Клієнтська частина*

Клієнтська частина системи моніторингу може бути зроблена у декількох формах:

1) Мобільний додаток.

   Дана форма клієнтської частини дозволяє використовувати функціонал навіть без персонального комп'ютера, достатньо лише телефон. Але, смартфонам історично бракує комфортної можливості роботи з файлами. Так як дана система потребує завантаження медіа контенту заради, наприклад, пошуку, то клієнтська частина у вигляді додатку до смартфона буде не зручною для використання.

2) Застосунок для персонального комп'ютера.

   Програма, що працює на персональному комп'ютері працює найшвидше. Але існує проблема встановлення цього програмного забезпечення. Більшість інструментів розробки вимагають від клієнтського комп'ютера певного програмного забезпечення, як Python або .Net Framework. Також, вони потребують певної швидкодії від нього. Таким чином, клієнтська частина на базі застосунку для персонального комп'ютера дуже залежить від системи користувача, що є великим недоліком, адже дуже складно передбачити яку саме систему має кінцевий користувач.

3) Веб-застосунок.

   Даний тип клієнтського застосунку найбільш гнучкий, адже кожен комп'ютер має веб браузер, а складні обчислення можуть робитися на сервері. Веб-застосунок покаладає найменші вимоги на систему користувача, тому є дуже привабливою платформою для розробників.

Також, він не потребує жодних маніпуляцій для використання як, наприклад, встановлення програмного забезпечення і є кросплатформеним.

Отже, переглянувши всі варіанти, було обрано клієнтську частину у вигляді веб застосунку.

На даний момент обов'язковими технологіями для веб розробки є мова розмітки HTML та мова програмування JavaScript. Для розробки логіки відображення, програмісти використовують фреймворки, що спрощують додавання інтерактивності до розмітки сторінки. Найвідоміші фреймворки – Angular від компанії Google, React від компанії Facebook та Vue. Перші два широко використовуються в індустрії для великих проектів. Через те, що даний застосунок не є таким великим за обсягом, що над ним працюють декілька команд, а розробником є лише одна людина, можна обрати простішу за використанням технологію. Для невеликих проектів Vue дозволяє дуже швидко прототипувати та розвивати програмний застосунок. Обравши швидкість розробки понад технічні можливості фреймворку, оберемо Vue.

2.2.2. *Серверна частина*

Однією із веб популярних технологій є ASP.NET. Нещодавно, компанія Microsoft випустила нову версію популярного фреймворку та назвала її ASP.NET Core. Ця версія за даними Techempower [9] є однією з найшвидших з усіх технологій веб розробки. Точніше, посідає 7 місце з 307 по кількості текстових запитів у секунду Для порівняння, технологія Java Spring посідає 179 місце, а python django – 185. Також, ASP.NET Core з веб сервером Kestrel підтримує останній стандарт HTTP 2.0.

Для розгортання потрібно обрати операційну систему. Звісно, потім її можна буде змінити, адже даний застосунок в ядрі буде використовувати тільки кросплатформені інструменти та продукти. Переглянувши ціни хостингу на ОС Windows та Linux, помітимо, що обслуговування останньої

набагато дешевше [10] (див. табл. 2.1), ОС Linux надає більшу можливість для налаштувань. Отже, оберемо ОС Linux.

Таблиця 2.1

Ціни на машини із різними ОС у провайдера Amazon

ОС	Пам’ять (ГБ)	Ядер процесора	Ціна за місяць ($)
Windows	0.5	1	5.93
Linux	0.5	1	4.25
Windows	3.75	2	140.55
Linux	3.75	2	73.20

У серверної частини є необхідність в комунікації із прикладним інтерфейсом Reddit. Для цього можна написати свій інструмент доступу до неї, але доцільніше використовувати вже готові бібліотеки.

Першою такою можливою бібліотекою може бути бібліотека з назвою RedditSharp [11]. Вона має більше 150 зірок на популярній платформі хостингу контроля версій GitHub. Ця бібліотека була скачана близько 25,000 разів, тому можна підсумувати, що вона є досить популярною.

При тестуванні функціональних можливостей моніторингу Reddit було визначено, що ця бібліотека близько в 10 разів повільніша за альтернативну бібліотеку для мови програмування Python – PRAW [12]. Після аналізу, була знайдена проблема в тому, що реалізація бібліотеки RedditSharp не підтримувала посторінкове читання публікацій. Замість того, що прочитати декілька публікацій на сторонці, бібліотека робила додаткові запити на кожну.

Вихідний код бібліотеки RedditSharp був у відкритому доступі, тому можна було вирішити проблему. Автору бібліотеки було запропоноване рішення цієї проблеми за допомогою механізму pull request. Даний механізм

системи контролю версій дозволяє надавати автору бібліотеки можливість приймати зміни в вихідному коді від інших людей. Після огляду коду, автор погодився із запропонованими змінами [13]. При подальшому тестуванні було підтверджено, що ця проблема продуктивності зникла.

Далі були проведені тести довготривалого моніторингу соціальної мережі. Під час їх проведення були виявлена ще одна проблема цієї бібліотеки. Проблема полягала в тому, що при моніторингу Reddit більше години, продуктивність знаходження публікацій значно зменшувалася: до 1 публікації в 30 секунд.

Через велику кількість проблем з бібліотекою RedditSharp, було вирішено шукати альтернативу. Після повторного аналізу була знайдена бібліотека Reddit.NET [14], яка не мало вище описаних проблем, тому було вирішено використовувати її.

2.2.3. *Хмарний провайдер*

Через технічну вимогу до горизонтального масштабування доцільно використовувати хмарні технології. Такі технології надають декілька провайдерів: Google Cloud, Amazon Web Services, Azure. Для першої версії даного продукту не потрібні надвеликі обчислювальні потужності, тому в першу чергу можна переглянути безоплатні можливості даних продуктів. В цій категорії найгірше себе показує Microsoft Azure, адже більшість безоплатних сервісів доступна лише 12 місяців з дати реєстрації [15]. AWS та Google Cloud надають так звані «завжди безоплатні» послуги для сервісів, такі як AWS Lambda, Google Cloud Function, AWS Simple Queue Service, Google Cloud Pub/Sub та інші.

Також візьмемо до уваги, що для системи моніторингу потрібно розробити систему авторизації та реєстрації. Досить привабливим є те, що Amazon Web Services надають таку хмарну послугу. Сервіс, що надає такі можливості називається AWS Cognito. Google Cloud має схожий за

функціоналом Google Cloud Firebase, але ця послуга націлена в основному на мобільні девайси.

Переглянувши усі варіанти, можна зробити висновок, що найзручнішим провайдером є AWS, або Amazon Web Services. Необхідно зауважити, що система повинна проектуватись таким чином, щоб за нагоди можна було замінити провайдера на інший, наприклад, Google Cloud.

2.3. Архітектура системи

Дана система складається з декількох компонент, що можуть розгортатися незалежно. Також, зміни до них можуть вноситись незалежно. Ці компоненти дуже тісно співпрацюють із сервісами хмарного провайдера. Для схеми архітектури див. рис. 2.1. Виділимо всі компоненти системи на схемі:

1) Клієнтська частина web застосунку (статичні файли).
2) Веб сервер для доступу до даних.
3) Процес, що моніторить всі нові пости з картинками соціальної мережі Reddit.
4) Процес-обробник зображень, що перетворює зображення для зручності та швидкості пошуку.
5) База даних з метаданими зображень.
6) Система авторизації.

Система складається з двох принципових частин:

1) Підсистема відображення та пошуку (зверху на схемі).
2) Підсистема збору та обробки даних (знизу на схемі).

Розглянемо спочатку підсистему, з якою взаємодіє користувач, ту яка відповідальна за відображення збережених даних.

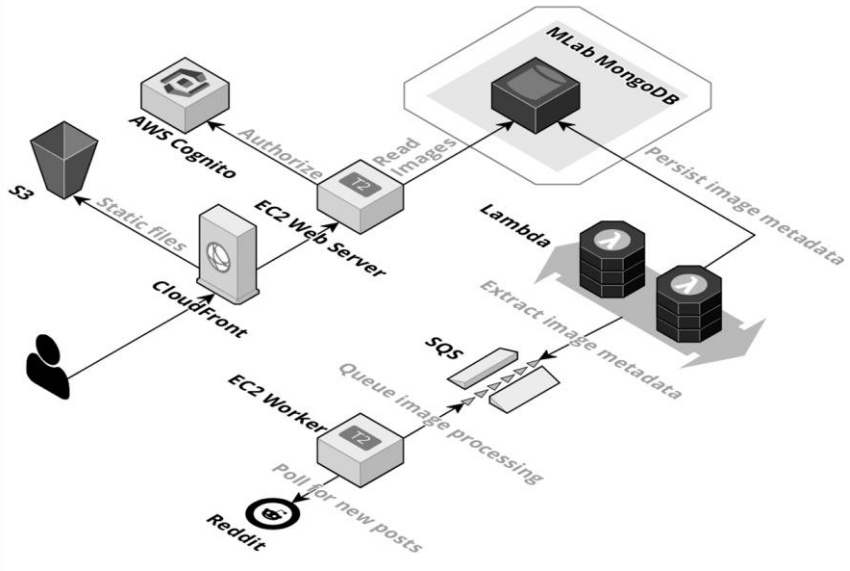


Рис. 2.1. Схема архітектури застосунку

2.3.1. *Підсистема відображення та пошуку*

Завдання підсистеми відображення — на основі даних підсистеми збору даних, підготувати результати пошуку. Також, до відповідальності даної підсистеми входять такі базові речі, як кешування статичного контенту та авторизація користувачів.

1) *AWS CloudFront.*

Перше, з чим стикається веб браузер користувача під час переходу на веб сайт застосунку – система AWS CloudFront. Ця система – CDN (Content Delivery Network), або мережа доставки контенту.В даній схемі вона віддає статичні файли, або направляє до веб серверу в залежності від запиту веб браузера.

Статичні файли дуже вигідно доставляюти через CloudFront, адже ця система надає послугу кешування. Кешування здійснюється за допомогою великої кількості серверів – Edge Locations [16]. Ці сервери розташовані у різних комірках світу. Якщо людина поряд з цим сервером робить запит на статичний контент, то перший раз контент проходить весь шлях від серверу хостингу цього контенту.

Edge Location віддасть контент та збереже локальну копію даних у своїй файловій системі. Тому при наступному запиті з того ж місця цей сервер віддасть контент без затримки для завантаження цього контенту з далекого серверу хостування.

Наприклад, користувач знаходиться в Китаї, а сервер хостування – в США. Edge Node в Китаї на другий запит не буде робити завантаження з серверу в США, а віддасть з локального диску. Отже, затримка буде складатися лише з часу походу на сервер в Китаї.

2) *Amazon S3*.

Статичні файли клієнтської частини зберігається на сервісі Amazon S3 (Simple Storage Service). Він надає гнучку можливість завантажувати файли та легко взаємодіяти з CloudFront. Для S3 на CloudFront налаштоване стискання та кешування для пришвидшення доставки веб застосунку до користувача.

3) *EC2 Web Server*.

Для безпечної даних між базою даних та клієнтським застосунком повинен бути шар серверного застосунку, який виконує операції для яких потрібно маніпулювати базою даних та зовнішніми сервісами.

В даній архітектурі цей шар представлений віртуальною машиною EC2 з серверною частиною запущенною на цій машині. Веб сервер відповадає за комунікацію з базою даних. Перед кожною операцією з даними цей модуль впевнюється в правах користувача, або авторизує його за допомогою зовнішнього сервіса.

4) *AWS Cognito*.

Описаний вище сервіс авторизації – AWS Cognito, який надає у вигляді сервіса наступний функціонал. По-перше, за допомогою нього можна створювати, або реєструвати користувачів. Після створення користувача сервіс надає можливість підтвердити аккаунт за допомогою SMS або електронної пошти. Без підтверження реєстрації він забороняє користувачу виконувати будь-які операції.

Вся комунікація з цим модулем від Amazon виконається на базі відомих HTTP запитів на відповідей, що сильно спрощує інтеграцію.

5) *MLab Mongodb*.

Майже кожен застосунок потребує місця для збереження та зчитування даних. В даному випадку, кількість картинок може бути величезною та швидкість їх вставки не є малою. З такими вимогами потрібно обрати базу даних. Також, від бази даних необхідні інструменти для роботи з великою кількістю даних. Виходячи з вище сказаного, було вирішено обрати NoSQL базу, а саме MongoDB.

MongoDB містить вбудовані інструменти реплікації, що дозволить збільшити кількість машин при збільшенні кількості даних. Також, вона містить так званий Aggregation Pipeline — потужний спосіб оброблювати величезну кількість даних. Це можна використовувати при побудові алгоритму пошуку.

Наразі одним із провайдерів безкоштовного сервісу використання MongoDB є MLab. При збільшенні навантаження завжди є можливість використати іншого провайдера, або перейти на платний план. Також, одним із варіантів є міграція на сервіс Amazon DocumentDB, що надає інтерфейс ідентичний MongoDB.

EC2 Web Server використовує дані з цієї бази даних для відображення результатів пошуку.

Вище був повністю описана архітектура частини, що відповідає за зображення даних. Перейдемо до іншої, не менш важливої підсистеми застосунку.

2.3.2. *Підсистема збору та обробки даних*

Дана підсистема відповідає за збір всіх релевантних даних з соціальної мережі Reddit та трансформації її у вид, що дозволяє ефективно шукати по ним.

1) *EC2 Worker*.

До Reddit кожної секунди надходять різні публікації, для їх обробки потрібно якось ці публікації відслідковувати. Для цього було вирішено використовувати процес-worker на віртуальній машині у Amazon. Worker (робітник) – процес, що виконує певну роботи незалежно від сигналів зовнішнього світу. На відміну від веб серверу, такий робітник не обробляє запити, а працює завжди.

В даній архітектурі відповідальність цього worker-процесу, робити poll-запити кожен певний проміжок часу та перевіряти чи були надіслані нові публікації. При успішному запиті на нові публікації, процес повинен обробити текст та виділити усі посилання на зображення. Кожне таке посилання повинне бути оброблене. Цей процес не може бути зайнятим іншою задачею на великий проміжок часу, адже якщо декілька секунд не перевіряти нові публікації, їх з’явиться дуже багато та буде неможливо за виділений час їх опрацювати. Тому, так як неможливо визначити наперед скільки таких зображень буде отримано та скільки часу знадобиться для їх обробки, задачу більш глибокої обробки зображення потрібно перекласти на інший процес з незалежними обчислювальними потужностями. Для того, щоб не втратити цю роботу на поставити її в «чергу», потрібно використати інший сервіс

2) *SQS*.

Amazon SQS – сервіс, що надає можливості розподіленої черги. Тобто за допомогою нього можна ставити роботу в чергу та виконувати її.

Так як ця черга розподілена, вона може приймати величезну кількість повідомлень та з немалою швидкістю ставити та забирати роботу з потоку. Однією з важливих можливостей такої черги є підтримка декількох спроб виконати роботу. Наприклад, якщо у сервера, на якому зберігається зображення є якісь перебої, то робота не може виконатися, тобто вона перейде до стану «збій». При переході в такий стан, SQS дає можливість виконати ще декілька спроб обробки

повідомлення. Кількість таких спроб задається конфігурацією. У випадку, коли після всіх спроб операцію не може бути виконана, ця роботу переходить до іншого сховища, де зберігаються всі повідомлення, які не можуть бути виконані. Таке сховище називається dead-letter queue [17].

3) *Lambda*.

Повідомлення з посиланнями на картинки в черзі SQS потрібно кимось оброблювати. Через можливу велику та нерівномірну кількість зображень, що можуть надходити з соціальної мережі Reddit потрібно обрати такий обробник, що автоматично горизонтально масштабується.

Amazon Lambda – сервіс, що призначений для обробки подій, повідомлень та інших одиниць роботи. Цей сервіс надає можливість інтеграції з Amazon SQS. Lambda динамічно реагує на повідомлення SQS та запускає процеси-обробники. Так як Lambda є розподіленим сервісом, такі обробники повідомлень можуть запускатися паралельно. Ця паралелізація конфігурується. Кількість процесів-обробників залежить від кількості повідомлень у черзі, що гарантує горизонтальну масштабованість.

Отже, такий підхід надає можливість обробити будь-яку кількість зображень, навіть при нерівномірному розподіленні публікацій у соціальній мережі.

Задача обробника Lambda полягає в скачуванні картинки по посиланню. Цей обробник також повинен виділити метадані для майбутнього пошуку. Цей процес буде розглянутий у наступній главі більш детально.

Після виділення метаданих, ці дані потрібно скласти в базу даних MongoDB для подальшого пошуку

2.4. Розгортання системи

Розгортання системи – дуже важливий процес, адже тільки він гарантує готовий продукт після виконання усіх дій. В розгортання входить:

1) Конфігурація машин.
2) Конфігурація підсистем від інших організацій, з якими комунікує застосунок.
3) Компіляція вихідного коду.
4) Розміщення скомпільованого застосунку на машині розгортання.
5) Запуск розміщеного застосунку із потрібною конфігурацією.

Системою розгортання було обрано Azure Pipelines, так як вона безкоштовна та надає велику кількість можливостей без багатьох додаткових налаштувань.

Одним прикладом такої можливості, є виконання робіт для розгортання на машинах Linux Ubuntu 16.04. Це дозволяє бути впевненим в середовищі, яке буде використовуватися для розгортання та уникнути несподіванок.

Так як веб сервер та процес-worker будуть працювати на віртуальній машині EC2, також необхідна система, що буде конфігурувати стандартну віртуальну машину. Таким інструментом було обрано Ansible, який на даний момент є дуже популярним з величезними можливостями.

Поділемо розгортання системи на розгортання трьох підсистем:

1) Розгортання веб серверу із процесом збору зображень.
2) Розгортання клієнтського веб застосунку.
3) Розгортання процесу обробнику зображень.

2.4.1. *Розгортання веб серверу із процесом збору зображень*

Даний веб сервер буде розгортатися на машині з операційною системою на базі Linux. Так як прямого фізичного доступу до цієї машини надано бути не може, потрібно використовувати інструменти віддаленого доступу. Для таких операційних систем в індустрії використовують

технологію SSH (Secure Shell Access). SSH потрібно конфігурувати згенерувавши приватний та публічні ключі доступу. Завантажмо ці ключи на віртуальну машину у налаштуваннях створення. А на сервері розгортання використаємо цей ключ за допомогою задачі Install SSH Key в Azure Pipelines.

Завантаживши ключ, потрібно переконатися, що на машині розгортання встановлена остання версія Ansible. Переконавшись в цьому, згенеруємо файл налаштувань для майбутнього веб серверу. Конфігурація зберігається у файлі формату JSON (JavaScript Object Notation) з параметрами та секретними значеннями, такими як секретні ключі та паролі.

Наступним кроком нам потрібно запустити розгортаючий скрипт, написаний за допомогою Ansible.

Першим чином, цей скрипт встановить всі необхідні залежності на машині, на яку відбувається розгортання. В даному випадку, це – збірка runtime для asp.net core. Її встановлення проходить доволі легко, адже Microsoft надає усі версії збірки виконання у відкритому доступі.

Далі, він збудує застосунок та скопіює на потрібну машину. Також, для управління вже запущеним процесом дуже зручно користуватися systemd. Наприклад, цей менеджер процесів надає можливість автоматично перезапускати процес при невідомому збої. Згенеруємо файл налаштувань для systemd та надамо йому команду запустити серверний процес.

Для зручності розгортання, було вирішено розгортати веб серер та функцію збору зображень в одному процесі, тому при виконанні кроків вище автоматично запуститься збір зображень.

2.4.2. *Розгортання клієнтського веб застосунку*

Для клієнтського веб-застосунку використовується багато залежностей, їх потрібно встановити на машині розгортання для того, щоб мати можливість зібрати застосунок. Для цього використаємо NPM (Node

Package Manager). Через велику кількість залежностей, цей шаг є одним із найдовших в процесі розгортання.

Після встановлення необхідних залежностей, за допомогою них потрібно зібрати застосунок. Далі, достатнmo скопіювати вихідні файли від збірки до s3 bucket для статичних файлів. Цей s3 bucket повинен бути прив'язаним до CloudFront distribution заздалегідь для кешування

Для спрощення розгортання, перезапишемо налаштування кешування на index.html -–файл, який веб браузер завантажує першим. Нові налаштування повинні заборонити кешувати цей файл, щоб нова версія застосунку з'являлась одразу після завантаження збірки на s3 bucket, а не після того, як пройде період кешування

2.4.3. *Розгортання процесу обробнику зображень*

Для розгортання Lambda-функцій Amazon надає утіліту dotnet global tool, яку можна встановити за допомогою Nuget, що значно спрощує процес розгортання. Така утілита називається Amazon.Lambda.Tools. Dotnet global tool -залежність, що в середині містить звичайний консольний застосунок. В даному випадку, цей консольний застосунок займається всіма необхідними комунікаціями з сервісами Amazon для розгортання функції, що обробляє зображення.

Весь процес розгортки даного компонента складається із створення файла із налаштуваннями, що містить інформацію про різні обмеження функції, як пам'ять та максимальний час на обробку одного повідомлення та запуску Amazon.Lambda.Tools.

2.5. Висновки

В цьому розділі були розглянуті основні бізнес вимоги до проекту зі сторони автору публікацій, користувача, що проглядає публікації та модератору тематичного підсайту Reddit.

Також, наступним кроком були визначені головні технічні вимоги, що включають масштабованість, кросплатформеність та авторизацію користувачів.

Були обрана форма реалізації клієнтської частини у вигляді веб застосунку за допомогою мови розмітки HTML, мови програмування JavaScript та фреймворку VueJS. Щодо серверної частини, то найкращим інструментарієм для задачі опинилася мова програмування C# із фреймворком ASP.Net Core. Через вимогу масштабованості, було обрано хмарного провайдера Amazon для серверної частини.

На основі вимог та інструментів вище була побудована верхньорівнева архітектура застосунку, яка складається з двох логічних частин: підсистема зображення та пошуку і підсистема збору та обробки даних. Для реалізації цієї архітектури, було розроблено процес розгортання усіх компонент.

3. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ

3.1. Алгоритм виділення метаданих зображення

Найчастіше, зображення в пам’яті комп’ютера зберігаються у вигляді послідовності байт. Розглянемо один із найпопулярніших форматів збереження кольору – RGB. В цьому форматі колір кожного пікселя кодується за допомогою трьох основних складових:

1) Червоний колір (R).
2) Зелений колір (G).
3) Синій колір (B).

Кожна компонента може приймати значення від 0 до 255. Виходячи з цього, зображення може бути представлено и вигляді чисел. Тому два зображення можна порівняти за допомогою маніпулювання цих чисел.

Порівнювати два об’єкта найпростіше за допомогою однієї числової характеристики. Назвемо цю характеристику відстанню між двома зображеннями. Розглянемо декілька варіантів визначення відстані.

3.1.1. *Причини необхідності виділення метаданих*

Першим очевидним способом порівняти два зображення – це порівняти три кольорові компоненти усіх відповідних пікселів зображення з бази даних та об’єкту порівняння. Для такого підходу необхідно, щоб ці два зображення мали однакову роздільну здатність. Нехай n – кількість пікселів зображення. Символами *T* та *I* позначимо зображення. Позначимо відстань символом *S*. I_r^k – компонента *r* (червоний колір) пікселя *k* зображення *I*. Тоді за допомогою формули (3.1) можна знайти характеристику відстані між двома зображеннями, які мають однакову ширину та висоту, маючи дані про кожну кольорову компоненту кожного пікселя обох зображень.

$$d(I,T) = \sum_{k=1}^{n} \sqrt{(I_r^k - T_r^k)^2 + (I_g^k - T_g^k)^2 + (I_b^k - T_b^k)^2} \qquad (3.1)$$

Недоліки такого алгоритму:

1) Для порівняння потрібно перевірити всі пікселя обох зображень.

   Уявимо, що ми порівнюємо малі зображення 400x300 пікселів та таких зображень в базі даних 1000 одиниць. З таким алгоритмом потрібно буде перевірити сумарно 2400 пікселів 1000 разів, тобто 2 400 000 операцій. Таким чином, навіть на малій виборці з дуже малими зображеннями, кількість операцій надзвичайно велика. Також, при збільшення розміру зображення, кількість операцій росте

2) Картинки повинні бути одного розміру.

   Тобто, для порівняння зображень різного розміру, їх потрібно масштабувати, що погано впливає на швидкість порівняння.

3) В базі даних потрібно зберігати всі пікселі картинки.

   Через те, що для пошуку потрібно використовувати усі пікселі зображення, база даних потрібна містити всю інформацію про картинку. Це суттєво збільшує розмір бази даних та кошти на її підтримку

Виходячи з цих недоліків, потрібно розробити алгоритм, що порівняє картинки різних розмірів та швидкодія якого не залежить від розміру. Щоб досягти незалежності швидкодії пошуку від розміру алгоритму, потрібно виділити певні метадані зображення, або їх ще можна назвати певним зліпком цього зображення.

3.1.2. *Гістограма кольорів, як зліпок зображення*

Для цього, перед виконанням операції пошуку, потрібно обробити картинки таким чином, що вся потрібна для порівняння інформація була меншого розміру. Тобто, знайти корисний та зменшений зліпок вихідного зображення.

Замість використання кольору кожного конкретного пікселю, можна стиснути цю інформацію та використовувати загальне використання певного кольору у зображенні. А потім, шукати зображення зі схожим

розподілом кожного кольору. Для обрання формату такого зліпку зображення, потрібно вирішити як обрати найважливіші кольорові характеристики.

Рішення цієї проблеми побудуємо на основі декількох публікацій, що описують порівняння зображень на основі даних кольорових гістограм червоного, синього та зеленого кольорів [18, 19].

Розглянемо побудову гістограми на прикладі однієї компоненти – червоного кольору. Розділимо всі можливі значення 0-255 компоненти червоного кольору на чотири інтервали рівної довжини. Отримаємо такі інтервали: 0-63, 64-127, 128-191, 192-255. Дані інтервали – значення на горизонтальній вісі графіку. Після цього, кожен піксель потрібно розмістити на одному з цих інтервалів, що надасть розподіл різних значень кольорової компоненти на зображенні.

Розглянемо цей алгоритм на прикладі зображення 4x4 на рис. 3.1, кожен піксель якого має різну інтенсивність.

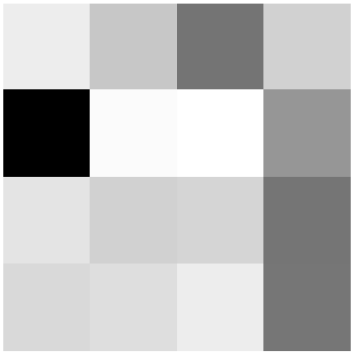


Рис. 3.1. Приклад зображення 4x4

Перевіримо всі пікселі даного зображення та випишемо значення червоної компоненти зображення для кожного пікселю. Див. результати у табл. 3.1. Кожне значення в даній таблиці відповідає значенню червоної

кольорової компоненти пікселя, що знаходиться у відповідний позиції на зображенні.

Таблиця 3.1

Значення червоного кольору картинки 4x4

45	89	156	78
245	19	12	128
56	78	74	155
70	63	45	154

З першого погляду, колір не є насиченим, але це потрібно довести за допомогою графіку. За допомогою алгоритму, що наведений вище, розподілимо всі значення з табл. 3.1 по інтервалам 0-63, 64-127, 128-191 та 192-255. З цього отримаємо нормалізований та звичайний розподіл, який наведений на рис. 3.2, який доводить припущення щодо малої насиченості кольору у зображенні.

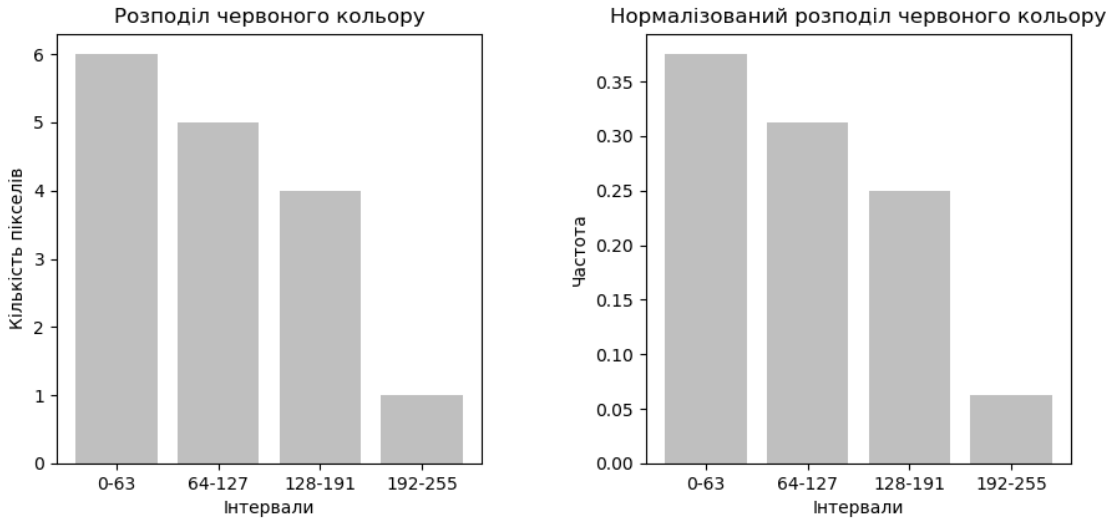


Рис. 3.2. Розподіл червоного кольору на зображенні

Клас на ліст. 3.1, що реалізує обрахування такої гістограми, називається *Histogram*. Метод *Add* даного класу оцінює кольору компоненту

та обирає відповідний інтервал. Далі, інкрементує кількість пікселів, що входять до даного інтервалу. Після оцінки таким чином всіх пікселів зображення, остаточний розподіл можна отримати за допомогою методу *GetNormalizedBuckets*, що повертає нормалізоване значення розподілу пікселів у кожному з інтервалів.

Лістинг 3.1. Клас, що представляє одну гістограму зображення

```
public class Histogram
{
    public string Name { get; }
    private readonly double _minValue;
    private readonly double _maxValue;
    private readonly int[] _buckets;
    private readonly double _step;
    public Histogram(string name, int buckets, double minValue, double
maxValue)
    {
        Name = name;
        _minValue = minValue;
        _maxValue = maxValue;
        _buckets = new int[buckets];
        _step = (_maxValue - _minValue + 1) / _buckets.Length;
    }
    public void Add(double value)
    {
        if (value > _maxValue || value < _minValue)
            throw new ArgumentException($"Value {value} is out of
bounds");
        _buckets[GetValueIndex(value)]++;
    }
    public double[] GetNormalizedBuckets()
    {
        var total = _buckets.Sum();
        var normalized = _buckets.Select(b => b /
(double)total).ToArray();
        return normalized;
    }
    private int GetValueIndex(double value)
    {
```

```
            return (int)((value - _minValue) / _step);
        }
    }
```

Далі, у випадку з картинками RGB потрібно виконати вище описані операції 3 рази та побудувати 3 таких гістограми для кожного відповідного кольору.

Таким чином, можна використати таку гістограму замість усіх пікселів картинки для операцій пошуку. У випадку картинки 4x4 замість 16 значень потрібно зберігати лише 12 – значення частоти кожного кольору в кожному із інтервалів. Також, при збільшенні картинки не збільшується кількість даних, які потрібно зберігати для подальшого пошуку. Для зображення розміром 1024x768 кількість значень для зберігання залишається незмінною – потрібно зберігати лише 12 значень.

3.2. Алгоритм пошуку схожих зображень

3.2.1. *Визначення схожості між гістограмами зображень*

Ми вже визначили які потрібно зберігати дані для пошуку, тепер потрібно обрати функцію схожості між гістограмами зображень в базі даних та гістограмаю зображення, що надає користувач.

В простому рішенні було використано функцію відстані, але вона є складною для обрахування. Для простоти обчислень можна обрати метод перетинань [20] для порівняння гістограм.

Визначимо $H(I)_i$, як i інтервал гістограми зображення H. Далі використаємо формулу (4.1) для обчислення характеристики перетину двох гістограм.

$$d(I,T) = \sum_i \min\left(H(I)_i, H(T)_i\right) \tag{4.1}$$

Тобто, для кожного інтервалу потрібно взяти найменше значення з гістограм двох зображень, що порівнюються та просумувати такі значення.

Також існує альтернативний метод хі-квадрат [21], який описаний формулою (4.2).

$$d(I,T)=\frac{1}{2}\sum_i \frac{\left(H(I)_i - H(T)_i\right)^2}{H(I)_i + H(T)_i} \tag{4.2}$$

Але, по-перше, така формула забороняє нульові значення в обох гістограмах, тому що відстань неможливо знайти при нульовому знаменнику. В конкретному інтервалі може не існувати пікселів, тому цей недолік є суттєвим. Також, така формула є обчислювально складнішою, що погано впливає на швидкодію пошуку.

Отже, оберемо формулу перетинів для обчислення відстані між гістограмами.

3.2.2. *Формат запису в базі даних MongoDB*

Розглянемо в якому форматі записується інформація про зображення у базі даних. Так як MongoDB – нереляційна NoSQL база даних, то вся інформація зберігається не за допомогою таблиць та залежностей між ними, а у вигляді документів, що зберігаються в колекціях. Розглянемо приклад запису в базі даних:

Лістинг 3.2. Приклад запису в базі даних

```
{
    "_id" : "8993346e-e62f-4db0-9bec-48390035eee2",
    "ImageUrl" : "https://i.redd.it/um1d9078a9x21.jpg",
    "RedditId" : "bsxo6v",
    "Url" : "/r/xqcow/comments/bsxo6v/we_forgot/",
    "CreatedAt" : ISODate("2019-05-25T18:19:37.000Z"),
    "Ignore" : false,
    "Subreddit" : "xqcow",
    "FeatureBuckets" : {
        "red" : [
            0.658897058823529,
            0.192405024509804,
            0.0837469362745098,
```

```
            0.0649509803921569
        ],
        "green" : [
            0.787858455882353,
            0.135629595588235,
            0.061890318627451,
            0.0146216299019608
        ],
        "blue" : [
            0.797846200980392,
            0.122668504901961,
            0.0600260416666667,
            0.0194592524509804
        ]
    }
}
```


Розглянемо кожне поле в цьому об’єкті.

1) _id.

   Дане поле – унікальний ідентифікатор в базі даних. При зборі інформації та обробці зображення можуть виникати помилки. За архітектурою процесу, такі помилки обробляються за допомогою повторного виконання операції. Щоб повторне виконання було ідемпотентним, тобто кожна спроба повинна приводити до того самого результату, при ініціюванні операції цій кожній картинці видається свій ідентифікатор. Тому якщо картинка вже збережена в базі даних, такий підхід дозволяє уникнути дублікатів.

2) ImageUrl.

   Це – посилання на зображення в мережі Інтернет, що дозволить показати його на у користувацькому інтерфейсі

3) RedditId.

   Соціальна мережа Reddit надає кожній публікації певний ідентифікатор. При необхідності отримати будь-яку додаткову інформацію, потрібно використовувати цей ідентифікатор.

4) Url.

Дане поле містить посилання на публікацію в соціальній мережі. Для того, щоб користувач міг перейти до публікації, йому потрібно надати це посилання.

5) CreatedAt.

Дата створення публікації, що дозволить фільтрувати найрелевантніші публікації при необхідності.

6) Ignore.

Певні пости є автоматично створеними ботами, модераторами, або мають позначку NSFW. Для того, щоб фільтрувати такі пости, використовується цей маркер.

7) Subreddit.

Дане поле використовується у запитах для фільтрування публікацій за певними підрозділами, або сабредітами.

8) FeatureBuckets.

В цьому місці зберігається вся інформація, що необхідна для пошуку за зображенням. Поле має вигляд об’єкту, який зберігає інформацію по обробленим гістограмам для даного зображення. Маємо три гістограми: для червоного(red), зеленого(green) та синього(blue) кольорів. Інформація для гістограми зберігається у виді масиву значень для відповідних інтервалів кольорів: 0-63, 64-127, 128-191, 192-255.

3.2.3. *Алгоритм пошуку схожих зображень в базі MongoDB*

Дана NoSQL база даних має потужний інструмент пошуку, який називається Aggregation Pipeline. Даний механізм дозволяє комбінувати різні операції (які називаються етапами) у єдиний процес, що дозволяють виконувати складні математичні обчислення. Природа цього інструмента надає можливість виконувати операції над величезними об’ємами даних Він не має обмеження розміщення результату в оперативній системі, адже Aggregation Pipeline пітримує збереження частичних результатів на диску.

Отже, потрібно за допомогою механізмів Aggregation Pipeline розробити алгоритм порівняння зображень, що по своїй суті є алгоритмом порівняння гістограм

Спочатку, треба відфільтрувати зображення, які не потрібно використовувати для пошуку схожих. Це зображення, що ігноруються та також такі, що не були опубліковані в сабредітах, які обрав користувач. Для цього використаємо етап під назвою $match у якому зазначимо використовувати тільки такі записи, які у полі Ignore не мають значення true. Також, надамо інструкцію відфільтрувати усі записи, які у полі Subreddit мають значення, що не співпадають з налаштуваннями користувача:

Лістинг 3.3. Етап фільтрування

```
{
    $match:{
        Ignore:{
            '$in':[  null, false ]
        },
        Subreddit:{
            '$in':[
                'subreddit1',
                'subreddit2'
            ]
        }
    }
}
```

Далі, підготуємо дані для обчислень перетину гістограм. Візьмемо усі три гістограми зображення користувача та об’єднаємо в одну. Зробимо те саме для кожного зображення бази даних оператором $contactArrays. Далі транспонуємо ці два масиви таким чином, щоб на виході отримали по масиву на кожен інтервал кожного кольору. Даний елемент зображення користувача стане у відповідність такому у інтервалу зображення у базі даних. Наприклад, масиви [0.1, 0.6, 0.3] та [0.2, 0.4, 0.4] перетворяться на

масиви [0.1, 0.2], [0.6, 0.4], [0.3, 0.4]. Таку трансформаццію виконує оператор $zip. Додамо цю інформацію до кожного зображення в базі за допомогою етапу $addFields:

Лістинг 3.4. Етап підготовки даних для алгоритму порівняння гістограм

```
{
   $addFields:{
      allFeatures:{
         $zip:{
            inputs:[
               [
                  0.1,
                  0.2,
                  0.3,
                  0.4,
                  …
               ],
               {
                  $concatArrays:[
                     '$FeatureBuckets.red',
                     '$FeatureBuckets.green',
                     '$FeatureBuckets.blue'
                  ]
               }
            ]
         }
      }
   }
}
```

Тепер за алгоритмом потрібно просумувати усі мінімальні значення відповідних комбінацій гістограм. Так як агрегаційні функції MongoDB вміють працювати тільки з групами документів, розіб’ємо документ на піддокументи кожен з яких має тільки два значення із минулого етапу – одне з гістограми зображення користувача, друге відповідне значення із зображення бази даних. Для цього використаємо етап $unwind, що розбиває

масив на стільки документів, скільки значень існує в масиві. Потім згрупуємо по ідентифікатору ці документи та просумуємо мінімальні значення відповідних інтервалів гістограм:

Лістинг 3.5. Етап обрахування значення перетину гістограм

```
{
   $unwind:{
      path:'$allFeatures'
   }
},
{
   $group:{
      _id:'$_id',
      intersection:{
         $sum:{
            $min:'$allFeatures'
         }
      },
      image:{
         '$first':'$$ROOT'
      }
   }
}
```

Після виконання усіх етапів, що описані вище, в полі *intersection* буде записано значення перетину гістограм кольорів зображення користувача з певним зображенням в базі даних. Останнім етапом буде відсортувати результат за спаданням по цьому полю. Вихідним масивом буде список зображень по зменшенню оцінки схожості з зображенням користувача, що і потрібно було знайти

3.3. Система автентифікації за допомогою AWS Cognito

Для підтримки автентифікації, авторизації та реєстрації застосунок повинен якимось чином зберігати інформацію про цих людей. Це збільшує розмір бази даних та вимагає від розробників час на її підтримку. Також,

застосунок в цьому випадку починає зберігати персональні дані, що значно посилює вимоги до безпеки цих даних. Тому Amazon для вирішення таких проблем надає сервіс AWS Cognito.

Цей сервіс зберігає користувацькі дані на стороні серверів Amazon на реалізує стандартні процеси.

Для використання цих процесів, від розробників застосунку вимагається тільки створити та налаштувати так званий User Pool. Після цього реєстрація користувача проводиться наступним чином:

1) Користувач вводить ім’я користувача, адресу електронної пошти та пароль;
2) Застосунок відсилає цю інформацію сервісу Cognito;
3) Cognito оброблює та зберігає ці дані. Користувач з’являється в панелі управління Cognito в статусі UNCONFIRMED. В цьому статусі сервіс не даватиме можливість авторизуватися цьому користувачу;
4) Сервіс Cognito надсилає на електронну пошту лист з посиланням для підтвердження реєстрації. Як тільки користувач перейде по такому посиланню, відповідний аккаунт в панелі управління переходить в стан CONFIRMED. Також, адміністратор може вручну підтвердити аккаунт за допомогою можливостей панелі управління Cognito.

Для авторизації Cognito використовує JWT токени. При вводі користувачем імені користувача та паролю, застосунок передає цю інформацію Cognito. Якщо такий користувач існую та наданий пароль є вірним, сервіс у відповідь видає певний токен. Життя цього токену обмежено в часі. Далі, клієнтський застосунок повинен при кожному запиті за інформацією до серверу, передавати токен в HTTP заголовку Authorization. Якщо токен не є валідним, або час життя цього токену вийшов, сервер у відповідь надасть код помилки 401.

Така поведінка серверу досягається за допомогою вбудованих можливостей ASP.NET Core [22]. Кожен раз, коли приходить запит на сторінку, яка вимагає авторизацію, сервер перевіряє наявність токену у

відповідному HTTP заголовку. При наявності такого заголовку, він посилає запит на AWS Cognito. Сервіс перевіряє валідність токену та за умови успішної перевірки, у відповіді надає код операцій 200. При неуспішній перевірці, сервіс віддає код помилки, а сервер забороняє виконання операції.

Так як Cognito зберігає всю інформацію у своїй базі даних, взаємодія з цим сервером виконується за допомогою HTTP запитів, або ручної роботи в панелі управління. Цей сервіс надає наступні широкі можливості налаштування:

1) Гнучкі налаштування вимог до паролю;

   Можна вимагати числа, спеціальні символи, символи верхнього та нижнього регістрів. Також, можна не вимагати будь-яку з опцій, наведених вище

2) Підтвердження реєстрації;

   Підтверджувати реєстрацію можна за допомогою електронної пошти та повідомлення на мобільний телефон. І лист на пошту, і текст повідомлення можуть бути змінені для кожного застосунку, що дозволяє стилізувати їх відповідно до стилю застосунку.

3) Мульти-факторна автентифікація (MFA).

   Можна вимагати від користувачів використовувати MFA , або надати можливість використовувати цю послугу індивідуально. Cognito підтримує MFA за допомогою електронної пошти та номеру телефона.

3.4. Висновки

В другому розділі було детально проаналізовано алгоритм пошуку схожих зображень. Починаючи з дуже простого рішення, було розроблене більш складний, але ефективний алгоритм. Цей алгоритм складається з двох частин: виділення метаданих зображень за допомогою мови програмування C# та операції пошуку по метаданим за допомогою механізму Aggregation Pipeline бази даних MongoDB.

Також, було описано алгоритм авторизації користувачів за допомогою сервісу Amazon Cognito. Такий підхід дозволив не зберігати жодні користувацькі дані та використовувати готові алгоритми часто використовуваних операцій для реєстрацій, автентифікації та реєстрації.

4. АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Особливості реалізації системи

Під час реалізації проекту було виділено декілька програмних модулів

1) Клієнтська частина;
2) Процес, який комунікує із Reddit для обробки зображень у нових публікаціях;
3) Процес обробки зображень та виділення гістограм кольорів;
4) Сервер, що ініціює процес пошуку та надає клієнтській частині дані.

4.1.1. *Масштабування*

Під час аналізу цих модулів було виявлено те, що найбільш вразливими до проблем із швидкодією та потужністю серверів є база даних та процес обробки зображень.

Так як базою даних була обрана MongoDB, то вона дозволяє покращувати свою працездатність за допомогою двох видів масштабування:

1) Вертикальне масштабування;

   Даний вид масштабування є найбільш очевидним. Він передбачає заміну машини, на якій працює процес бази на більш потужний по різним критеріям. Найголовнішими критеріями в даному випадку є характеристики процесора та оперативної пам'яті.

2) Горизонтальне масштабування.

   За своєю суттю, MongoDB є добре масштабованою базою даних, адже вона дозволяє з легкістю додавати машини до існуючих за допомогою шардування. При швидкому рості об’єму даних це надає можливість збільшувати обчислювальну потужність без руйнівного впливу на існуючу конфігурацію, тільки розширюючи її.

Гнучкість налаштування швидкодії процесу обробки в процесі обробки зображень досягається за допомогою декількох механізмів.

По-перше, для реалізації цього модулю було використано концепцію розподіленої черги, використовуючи AWS SQS. Це означає, що навіть при публікації величезної кількості зображень, система не буде чекати поки всі такі зображення будуть оброблені. Команди на обробку будуть поставлені в чергу та через певний час будуть виконані.

По-друге, для обробки команд з розподіленої черги було використано концепцію serverless, або безсерверної обробки за допомогою AWS Lambda. Такий підхід дозволяє автоматично масштабувати кількість функцій, що виконуються незалежно, виходячи з попиту на обчислювальні потужності. Якщо на певний момент команд на обробку мала кількість, то і кількість одночасних виконань функції також буде обмежено. З іншого боку, при різкому збільшенні кількості зображень, кількість функції також буде змінюватися відповідно. Важливо помітити, що кількість не є єдиною можливою характеристикою, яку можна змінити. Пам’ять, яка виділяється одному процесу безсерверної функції-обробнику також є конфігуруємою. Тобто, якщо по певній причині стане потрібно обробляти зображення більшого розміру, можна змінити виділену функції пам’ять за допомогою однієї змінної в панелі управління.

4.1.2. *Кросплатформеність*

Проаналізуємо виділені вище модулі на предмет підтримки кросплатформеності.

Клієнтська частина побудована як веб застосунок, що працює в браузері. Так як це браузерний застосунок, тому він за своєю природою є кросплатформеним, тобто таким, що може виконуватися на різних браузерах в різних операційних системах.

Процес комунікації із Reddit створений за допомогою кросплатформеного фреймворку .Net Core та не використовує можливості, що специфічні до певної платформи. Все, що треба цьому процесу – доступ в інтернет та можливість виконувати HTTP запити та отримувати відповідь

від таких запитів. Отже, такий процес можна виконувати в будь-якій операційній системі.

Процес обробки зображень було спроектовано таким чином, щоб він отримував певні вхідні дані, в даному випадку команди на обробку та певним чином обробляв їх за допомогою бібліотек, які не прив'язані до платформи. На вихід при успішній обробці не подається нічого, а при помилці генерується виключна ситуація. Сам програмний код може виконуватися на будь-якій системі, але AWS Lambda надає тільки ОС Linux. Тобто, при потребі зміни платформи, потрібно буде змінювати інфраструктурну частину коду.

Серверна частина, основна задача якого авторизувати користувача та видавати дані з бази MongoDB, розроблена за допомогою фреймворку ASP.Net Core, який може працювати як і на системі Linux, так і на системі Windows.

Можна зробити висновок, що усі модулі системи є достатньо кросплатформеними. Це надає можливість при необхідності використовувати їх на інших операційних системах без значних змін у вихідному коді.

4.1.3. *Оптимізація функції обробки зображень*

Після розроблення алгоритму виділення метаданих зображень, була протестована швидкодія даної функції на зображенні з великою роздільною здатністю 4032x3024. Тестування виявило, що таке зображення функція оброблює більше секунди. Даний результат було визнано незадовільним.

Під час аналізу вихідного коду, були знайдені декілька значень, які обчислювалися надлишково. Тобто, такі значення можна було обчислити тільки один раз, а програма робила ці дії при обробці кожного пікселя. Прикладом такого значення є змінна _step в класі Histogram. Ця змінна означає розмір кожного інтервалу гістограми. Так як ліва та права границі не змінюються під час роботи алгоритму, а є константами 0 та 255, дане

значення можна обчислити перед початком алгоритму та використовувати збережене значення.

Після оптимізації було виміряно швидкодію алгоритму на зображення різного розміру та роздільної здатності. Результати наведені в табл. 4.1. Це було зроблено за допомогою бібліотеки BenchmarkDotNet [23], яка є надзвичайно корисною для вимірювання продуктивності функцій. Бібліотека дозволяє вимірювати і виділену пам’ять, і процесорний час, затрачений на виконання алгоритму.

Таблиця 4.1

Швидкодія функції виділення метаданих

Роздільна здатність зображення	Розмір зображення, КБ	Середній час обробки, мс
800x1086	101	19
1125x1500	74	45
2060x1920	765	92
4032x3024	2681	292

Виходячи з результатів тестування, швидкість роботи функції обробнику не залежить від розміру зображення. Ця характеристика залежить тільки від кількості пікселів, які потрібно переглянути. Даний висновок можна зробити, звернувши увагу на те, що при збільшенні роздільної здатності зображення в табл. 4.1 з 800x1086 до 1125x1500 та зменшенні розміру зображення з 101 КБ до 74 КБ, час обробки зображення збільшується приблизно в два рази. Порівнюючи результат для найбільшого зображення, можна помітити, що швидкодія алгоритму після оптимізації збільшилась в 3-4 рази.

4.1.4. *Кількість необхідної пам’яті для роботи функції обробки зображень*

Під час тестування системи, було з’ясовано що для найбільшої пропускної здатності, функція обробки зображень потребує 1гб пам’яті. Цей процес також може працювати і з меншими ресурсами, але в такому випадку ймовірність обробити картинку сильно зменшується, тобто кількість необроблених через помилки зображень кардинально збільшується. Підтвердимо це за допомогою метрик, які Amazon автоматично збирає за допомогою системи CloudWatch (див. табл. 4.2). Дана система збирає метрики з усіх сервісів. У випадку даного застосунку – це SQS, S3, CloudFront та Cognito. В цих метриках можна знайти час обробки певних операцій, частота помилок та інші корисні цифри.

Таблиця 4.2

Статистика виконань функції обробки зображень за 5 хвилин

Пам’ять	Успішно оброблено	Помилки обробки	Ймовірність успішної обробки
128	35	554	6%
256	178	570	24%
512	301	390	44%
768	476	230	67%
1024	579	210	73%
2048	565	90	86%

Побудуємо графік залежності виділеної пам’яті від кількості команд на обробку зображень, які закінчилися успішно та невдало. Такий графік дозволить нам виділити граничне значення пам’яті, після якого не відбувається приросту у пропускній здатності обробника.

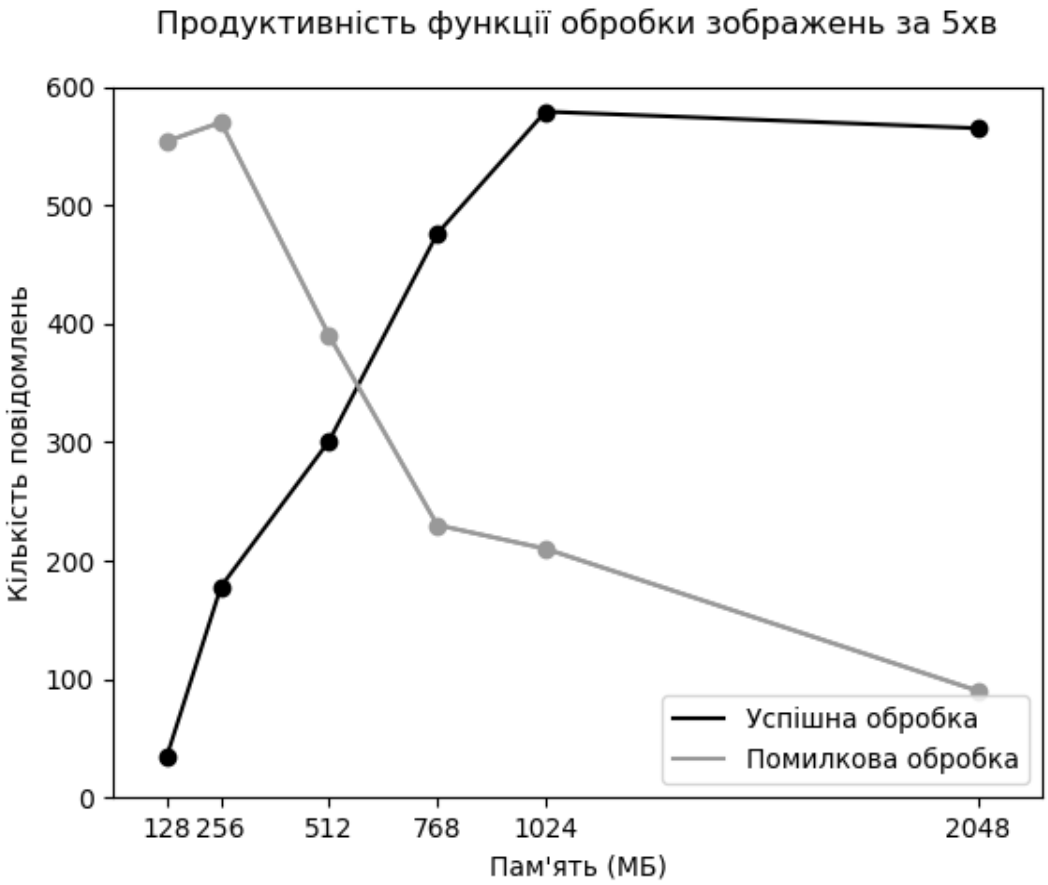


Рис. 4.1. Продуктивність функції обробки зображень в залежності від виділеної пам’яті

Виходячи з графічного відображення продуктивності функції на графіку на рис. 4.1, можна побачити що при збільшення пам’яті від 128 МБ до 1024 МБ значно впливає на продуктивність цієї функції. Але подальше збільшення пам’яті лише зменшує кількість помилок, а пропускна здатність, тобто скільки зображень функція встигає обробити, не змінюється.

4.1.5. *Зручність використання клієнтської частини*

Вся клієнтська частина кешується за допомогою CloudFront. Тому, хоча файли знаходиться на сервері в США, веб браузер в Україні отримує весь контент менше ніж за 100мс. По цій причині веб сайт завантажується дуже швидко, що дає користувачу можливість одразу почати використовувати важливі для нього функції.

Для почуття зворотнього зв’язку від застосунку до користувача, при розробці було вирішено додати так звані тост нотіфікації. Вони надають інформацію того, що певна операція завершилася за допомогою інформаційного повідомлення у верхній частині сторінки. Наприклад, якщо користувач ввів неправильні дані при авторизації, тост повідомлення миттєво надасть інформацію про це. Така сама поведінка використовується при реєстрації користувача, або якщо сесія користувача закінчилася.

Кожен HTTP запит спричиняє появу анімованої маски завантаження (див. рис. 4.2). Така маска приглушує задній фон та відображає обертаючийся елемент, який означає, що операція ще виконується. Як тільки сервер віддає певні дані, чи задовольняє певну команду, така маска зникає, сигналізуючи, що операція завершена. Такий підхід використовується на багатьох популярних веб застосунках, як twitter, youtube та ін.

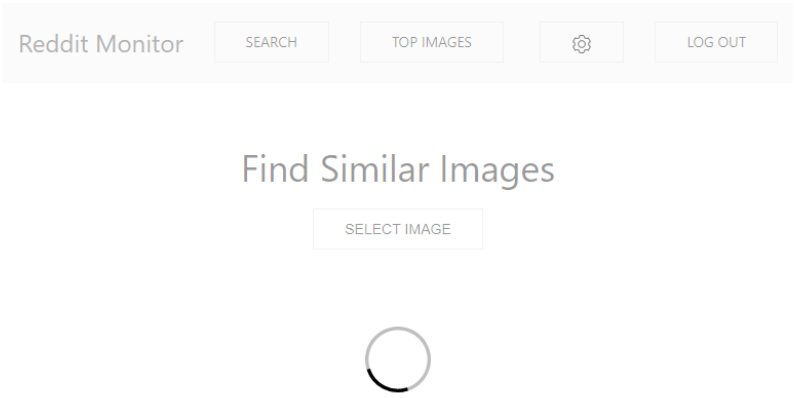


Рис. 4.2. Маска завнтаження

При інтеракції з певними елементами використовуються різні анімації. Прикладом такої анімації є перехід до списку публікацій, що мають таке саме зображення. При натисканні на кнопку для перегляду цих записів, виконується анімація перетворення одного елемента у список постів соціальної мережі.

Можна зробити висновок, що зручність використання була забезпечена за допомогою декількох технік, що широко використовуються на інших веб застосунках, навіть таких, які мають величезну аудиторію

4.2. Тестування системи

Для тестування потрібно обрати будь-яку картинку з пошукової системи. Оберемо зображення та опублікуємо його в тестовому сабредиті для пошуку (див. рис. 4.3).

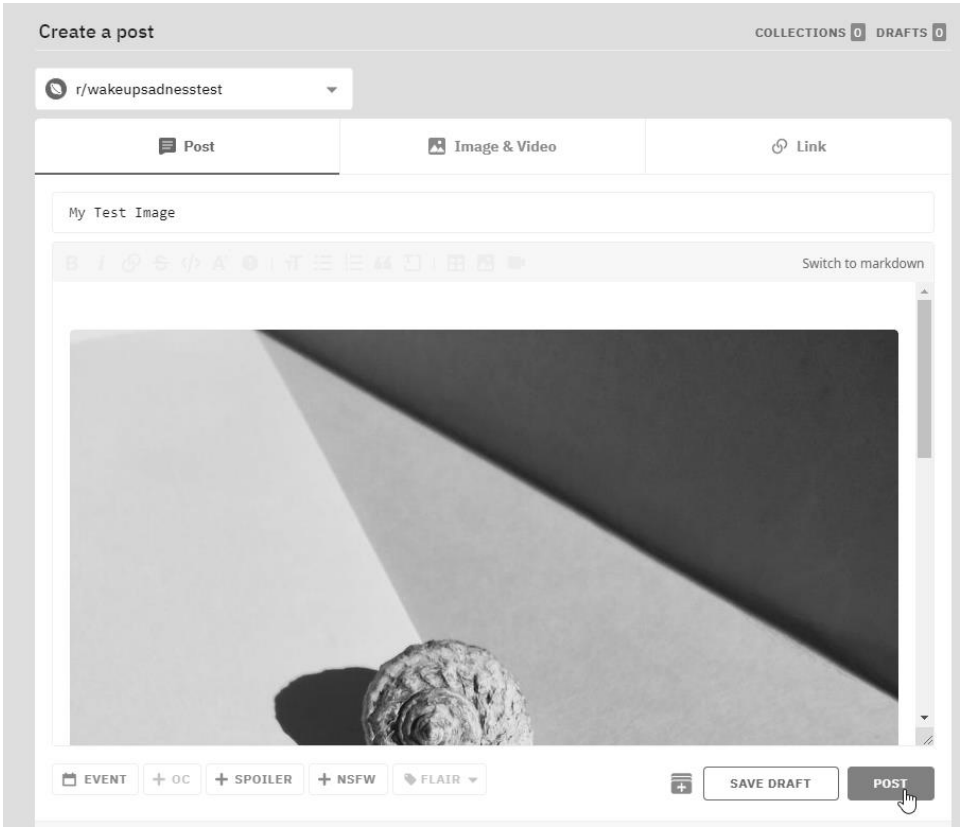


Рис. 4.3. Створення публікації із зображенням

Тепер змінимо картинку таким чином, щоб вона відрізнялася від опублікованої. Обернемо її на 90° та намалюємо декілька ліній:

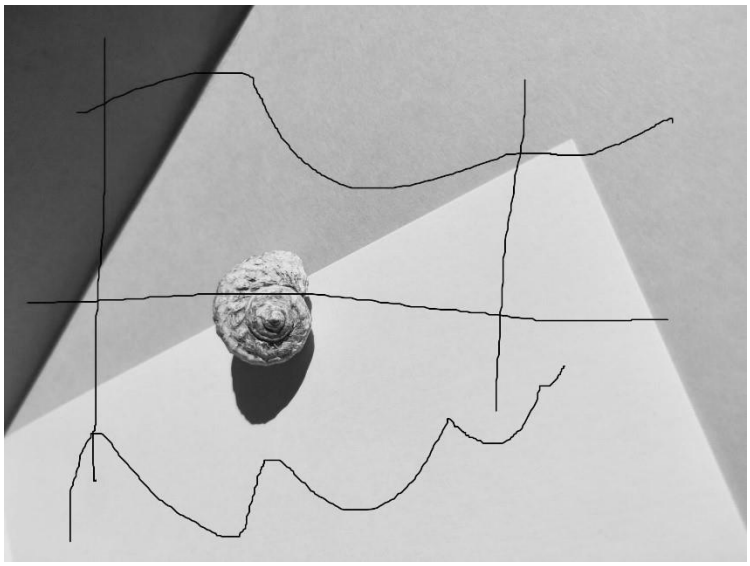


Рис. 4.4. Змінене зображення

Завантажуємо цю картинку та ініціюємо операцію пошуку. Отримаємо першим результатом шукану картинку (див. рис. 4.5). Отже, система збирає зображення в реальному часі та вміє шукати навіть за схожими зображеннями, що і треба було перевірити.

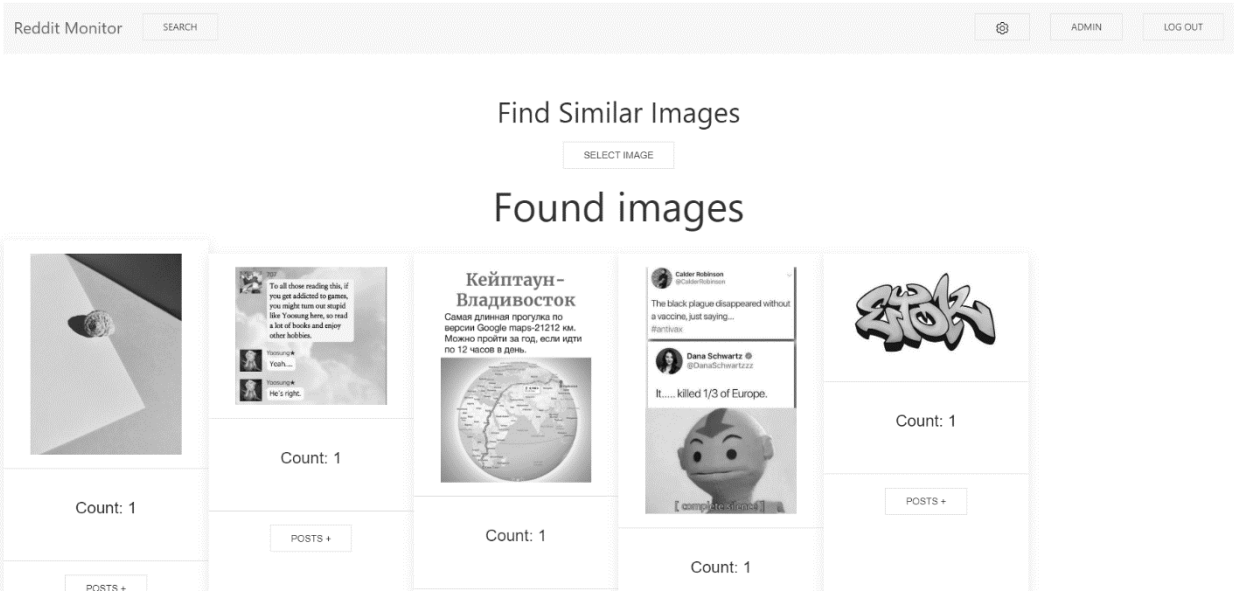


Рис. 4.5. Результати пошуку

4.3. Рекомендації щодо подальшого вдосконалення

4.3.1. *Алгоритм пошуку зображень*

В даному проекті було реалізовано порівняння зображень за допомогою кольорових гістограм. Даний спосіб є недостатньо ефективним при певних модифікаціях зображення: видалення об’єктів, зміні кольорової гами та ін.. Прикладом такої зміни є видалення людей з кадру фотографії природи. Але науковцями були знайдені інші характеристики [24], які покращують точність порівняння. Одна із рекомендацій для покращення ефективності роботи алгоритму – це впровадження декількох додаткових критеріїв порівняння:

1) Щільність краю.

   Для обчислення цієї величини, потрібно перетворити зображення на карту країв. Далі, для кожного пікселя, обчислити відношення країв до усіх пікселів у невеликий відстані до кожного. Під час порівняння зображень, різниця щільності країв і буде шуканою величиною для визначення схожості зображень

2) Текстурованість.

   Дана характеристика визначається кількістю пікселів, інтенсивність яких значно відрізняється від сусідніх. За допомогою неї можна виділити шаблони на зображенні.

3) Ранг.

   Ранг певного пікселю – кількість пікселів поряд з ним, інтенсивність яких менша за інтенсивність даного. Ранг може відобразити кольорові плями, які можуть мати семантичне значення. Ця характеристика надасть більшої ваги зображенням зі схожими кольоровими плямами.

Також, кольорова гістограма має певний недолік: при зміні кольорової гами зображення при порівнянні ця техніка зробить висновок, що вони не схожі. Для покращення роботи алгоритму пропонується дослідити підходи боротьби з такими проблемами. Одним із прикладів може бути додавання

ще однієї гістограми, яка є гістограмою зображення, перетвореного у відтінки сірого.

4.3.2. *Інфраструктура*

Інфраструктура застосунку може бути покращена за допомогою контейнеризіції серверного коду. За допомогою технології Docker можна ізолювати виконуючий код серверного застосунку та дозволити розгортати його за допомогою таких інструментів, як Kubernetes та ECS. Це в свою чергу додасть гнучкості до конфігурування серверних потужностей.

Нещодавно, Amazon запустив новий сервіс DocumentDB [25], який гарантує певну сумісність із протоколом MongoDB. Пропонується розглянути такий сервіс, так як він знаходиться в мережі Amazon та його використання зменшить ресурси, які витрачаються на комунікацію із базою даних. Також, компанія гарантує інтерфейси сумісності з іншими сервісами цього хмарного провайдера.

Ще однією рекомендацією є виділення процесу, який виконує запити за новими публікаціями соціальної мережі Reddit у самостійний процес, адже на даний момент він розгортається разом із веб сервером комунікації з базою даних. Це дозволить вносити зміни в ці два компоненти роздільно та розвивати їх паралельно, що значно прискорить швидкість розроблення у подальшому розвитку застосунку.

4.4. Висновки

В даному розділі було доведено, що розроблена система та її архітектура добре масштабується та є достатньо кросплатформеною. Перевіривши продуктивність функції обробки зображень, було зроблено оптимізацію коду, яку перевірено на різних вхідних зображеннях. Для цієї функції було встановлено оптимальну кількість виділеної пам’яті в 1 ГБ.

Протестувавши систему пошуку, було підтверджено функціональну можливість системи шукати схожі до зображення завантаженого користувачем.

З аналізу літератури було виділено таку рекомендації для підвищення якості пошуку: використовувати додаткові критерії порівняння зображень. Прикладом таких критеріїв є щільність краю, текстурованість та ранг. У випадку інфраструктури, було рекомендовано використати підхід контейнеризації застосунку та аналізу нової альтернативної бази даних DocumentDB.

ВИСНОВКИ

Метою даного дипломного проекту було розроблення системи моніторингу зображень в соціальній мережі Reddit. Даний застосунок було розроблено та протестована його основна функція.

Клієнтську частину цього проекту було реалізовано у вигляді веб застосунку за допомогою мови програмування JavaScript, мови розмітки сторінок HTML та фреймворку розроблення веб інтерфейсів VueJS. Серверна частина була побудована за допомогою мовою програмування C# із фреймворком .NET Core. Веб сервер використовує оптимізований фреймворк ASP.NET Core, що посідає 7 місце по швидкодії із 307 протестованих. Були використані хмарні сервіси від компанії Amazon.

Розроблена архітектура застосунку поділяє його на дві частини: підсистема відображення та пошуку і підсистема збору та обробки даних. Перша підсистема складається із веб сервера, клієнтської частини та інфраструктурними сервісами, що підтримують їх роботу. Друга підсистема складається із процесу збору публікації в соціальній мережі Reddit s процесу обробку зображень знайдених в цих публікаціях.

Під час реалізації цього дипломного проекту було розроблено алгоритм у мові C# для виділення характеристик зображень для подальшого пошуку на основі наукових публікацій. Особливістю цього алгоритму є мала кількість інформації навіть для великих зображень, що необхідна для подальшого пошуку. Також, було розроблено алгоритм пошуку за допомогою інструменту Aggregation Pipelines бази даних MongoDB.

Під час аналізу розробленого продукту було підтверджено його масштабованість та кросплатформеність. На основі оцінки продуктивності роботи функції обробки зображень було зроблено висновок у оптимальній виділеній пам’яті в 1ГБ.

Після аналізу було надані такі рекомендації для подальшого вдосконалення: впровадження додаткових критерії порівняння зображень,

використання технології Docker, розглядання нового сервісу DocumentDB як базу даних застосунку та виділення процесу збору публікацій в самостійний процес.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Reddit Statistics [Електронний ресурс] – Режим доступу: https://pushshift.io – (02.06.2019).
2. Karma Decay – Reverse image search of Reddit.com [Електронний ресурс] – Режим доступу: http://karmadecay.com/ – (02.06.2019)
3. Google Images – Режим доступу: https://images.google.com – (02.06.2019)
4. How Long Does It Take Google to Index a New Site [Електронний ресурс] – Режим доступу: https://www.theleverageway.com/blog/how-long-does-it-take-google-to-index-a-new-site/ – (02.06.2019)
5. Large Scale Online Learning of Image Similarity Through Ranking [Text] / Gal Chechik, Varun Sharma, Uri Shalit, Samy Bengio / Journal of Machine Learning Research – 2010 – Vol. 11. – P.538-543 - ISSN 1109-1135
6. Learning Fine-grained Image Similarity with Deep Ranking [Text] / Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, / The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) – 2014 – ISSN 1063-6919
7. Learning Cross-Spectral Similarity Measures with Deep Convolutional Neural Networks [Text] / Cristhian A. Aguilera, Francisco J. Aguilera, Angel D. Sappa, Cristhian Aguilera, Ricardo Toledo / IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) – 2016 – ISSN 2160-7516
8. A Novel Approach of Color Histogram Based Image Search/Retrieval [Text] / Apurva S. Gomashe, Prof. R. R. Keole / International Journal of Computer Science and Mobile Computing – 2015 – Vol. 4 № 6 – ISSN 230-088X
9. Round 17 Results – TechEmpower Framework Benchmarks [Електронний ресурс] – Режим доступу:

https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=plaintext – (02.06.2019)

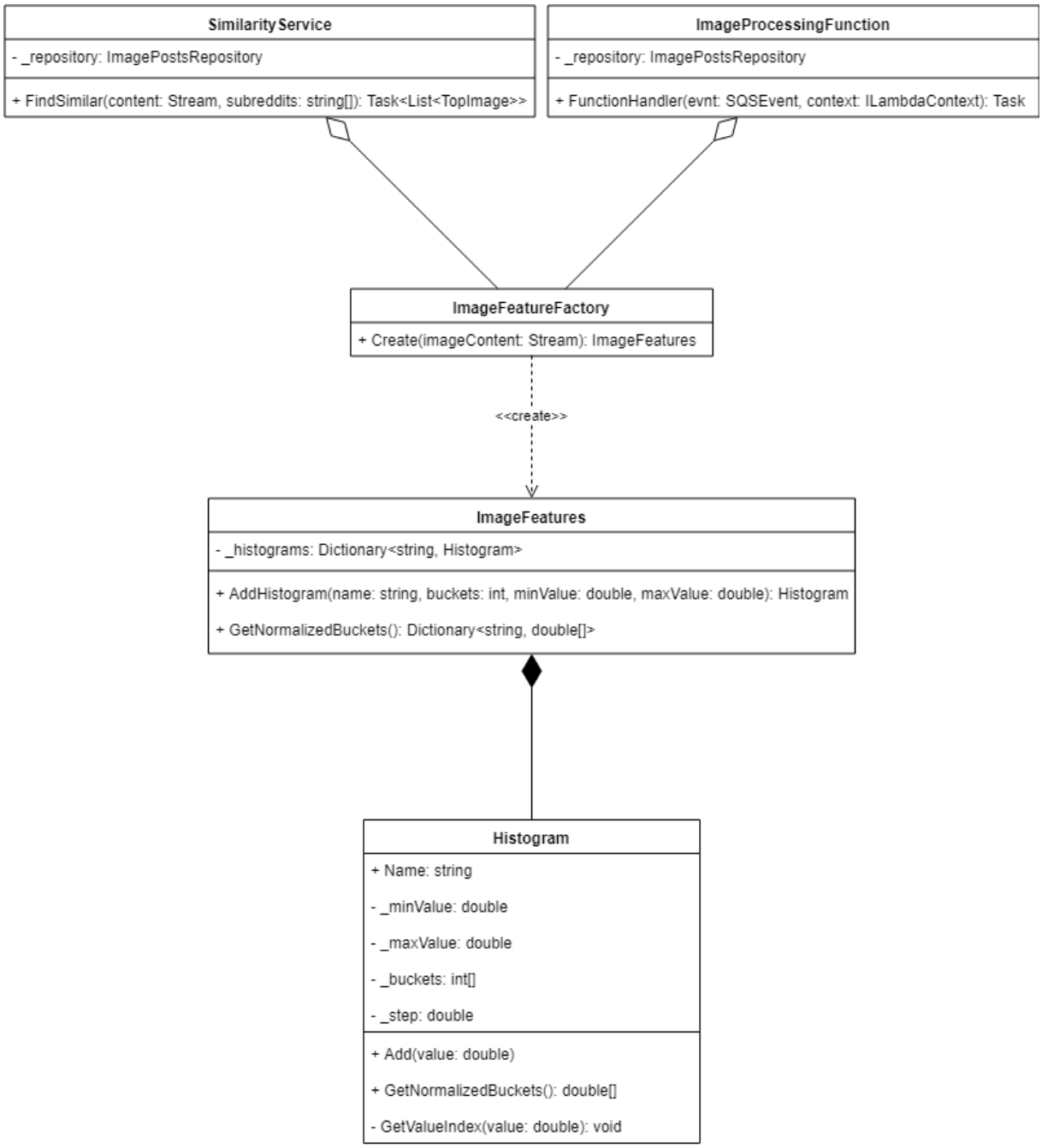
10. Amazon Web Services Simple Monthly Calculator [Електронний ресурс] – Режим доступу: https://calculator.s3.amazonaws.com/index.html – (02.06.2019)
11. CrustyJew/RedditSharp [Електронний ресурс] – Режим доступу: https://github.com/CrustyJew/RedditSharp – (03.06.2019)
12. PRAW: The Python Reddit API Wrapper [Електронний ресурс] – Режим доступу: https://praw.readthedocs.io/en/latest/ – (03.06.2019)
13. Fix Streaming Listing fetching new page for each next item by kostya9 [Електронний ресурс] – Режим доступу: https://github.com/CrustyJew/RedditSharp/pull/186 – (03.06.2019)
14. sirkris/Reddit.NET [Електронний ресурс] – Режим доступу: https://github.com/sirkris/Reddit.NET - (03.06.2019)
15. Carey S. Cloud vendor free tiers compared: AWS vs Azure vs Google Cloud Platform [Електронний ресурс] – Режим доступу: https://www.computerworlduk.com/cloud-computing/cloud-vendor-free-tiers-compared-aws-vs-azure-vs-google-cloud-3676667/ – (02.06.2019)
16. Key Features of a Content Delivery Network [Електронний ресурс] – Режим доступу: https://aws.amazon.com/cloudfront/features/ – (03.06.2019)
17. Amazon SQS Dead-Letter Queues [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html – (03.06.2019)
18. Image Similarity Measure using Color Histogram, Color Coherence Vector, and Sobel Method [Text] / Kalyan Roy, Joydeep Mukherjee / International Journal of Science and Research (IJSR) – 2013 – Vol. 2, №1. – P.538-543 – ISSN 2319-7064

19. A Review: Color Feature Extraction Methods for Content Based Image Retrieval [Text] / Divya Srivastava1, Rajesh Wadhvani and Manasi Gyanchandani / IJCEM International Journal of Computational Engineering & Management – 2015 – Vol. 18, №3. – P.10 – ISSN 2230-7893
20. Color indexing [Text] / M. Swain and D. Ballard. / International Journal of Computer Vision – 1991 – Vol. 7 №1 – P. 11-32.
21. Chi-Squared Distance Metric Learning for Histogram Data / Wei Yang, Luhui Xu, Xiaopan Chen, Fengbin Zheng, and Yang Liu / Mathematical Problems in Engineering – 2015 – Vol. 2015, Article ID 352849 – Режим доступу: https://doi.org/10.1155/2015/352849.
22. Barbettini N. Token Authentication in ASP.NET Core 2.0 – A Complete Guide [Електронний ресурс] – Режим доступу: https://developer.okta.com/blog/2018/03/23/token-authentication-aspnetcore-complete-guide – (03.06.2019)
23. Dotnet/BenchmarkDotNet [Електронний ресурс] – Режим доступу: https://github.com/dotnet/BenchmarkDotNet – (04.06.2019)
24. Comparing images using joint histograms / Greg Pass, Ramin Zabih / Multimedia Systems – 1999 – Vol. 7 №3 – P.234-240 – ISSN 0942-4962
25. Barr J. New – Amazon DocumentDB (with MongoDB Compatibility): Fast, Scalable, and Highly Available [Електронний ресурс] – Режим доступу: https://aws.amazon.com/blogs/aws/new-amazon-documentdb-with-mongodb-compatibility-fast-scalable-and-highly-available/ – (03.06.2019)

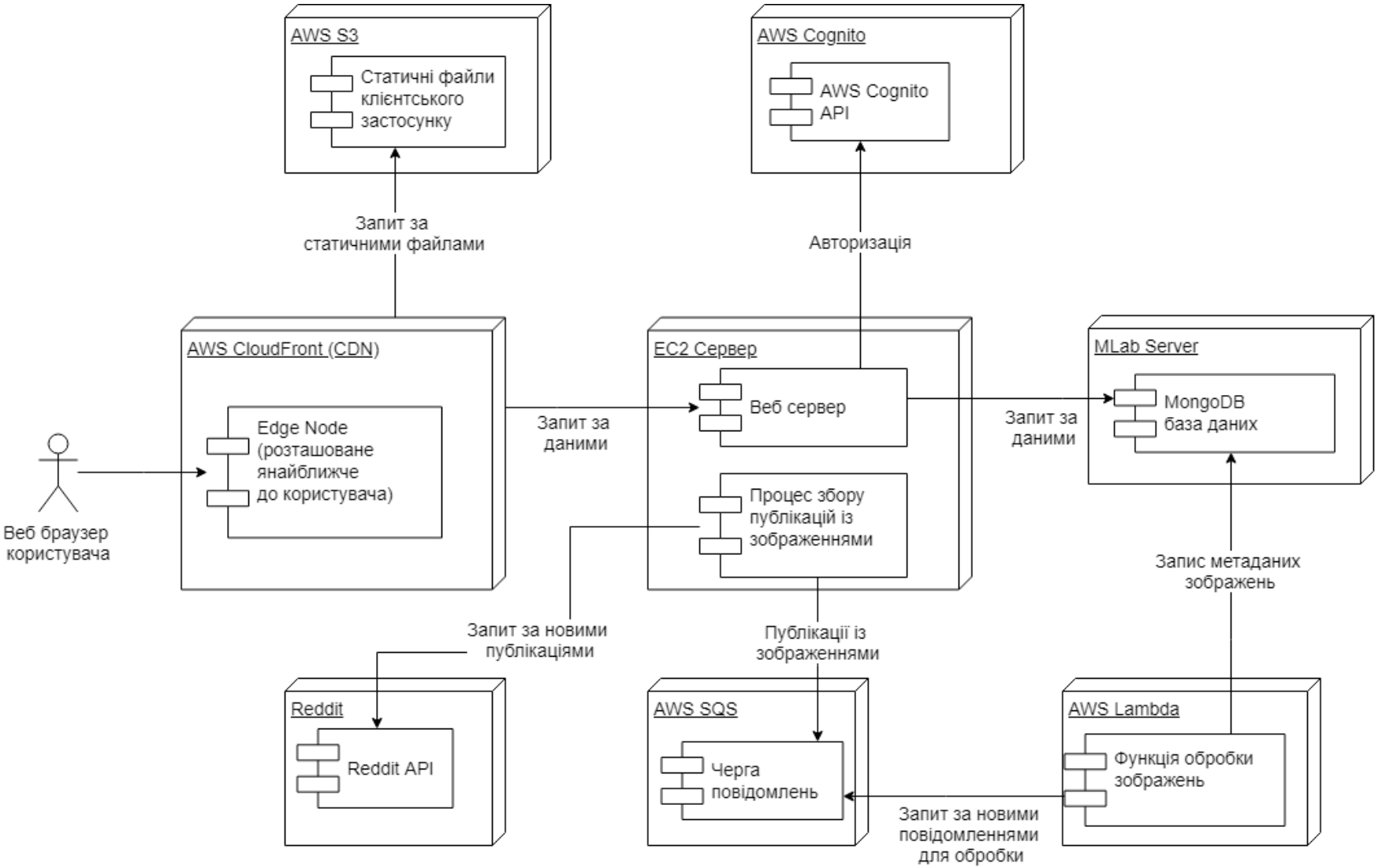
ДОДАТКИ

Додаток 1

Копії графічних матеріалів



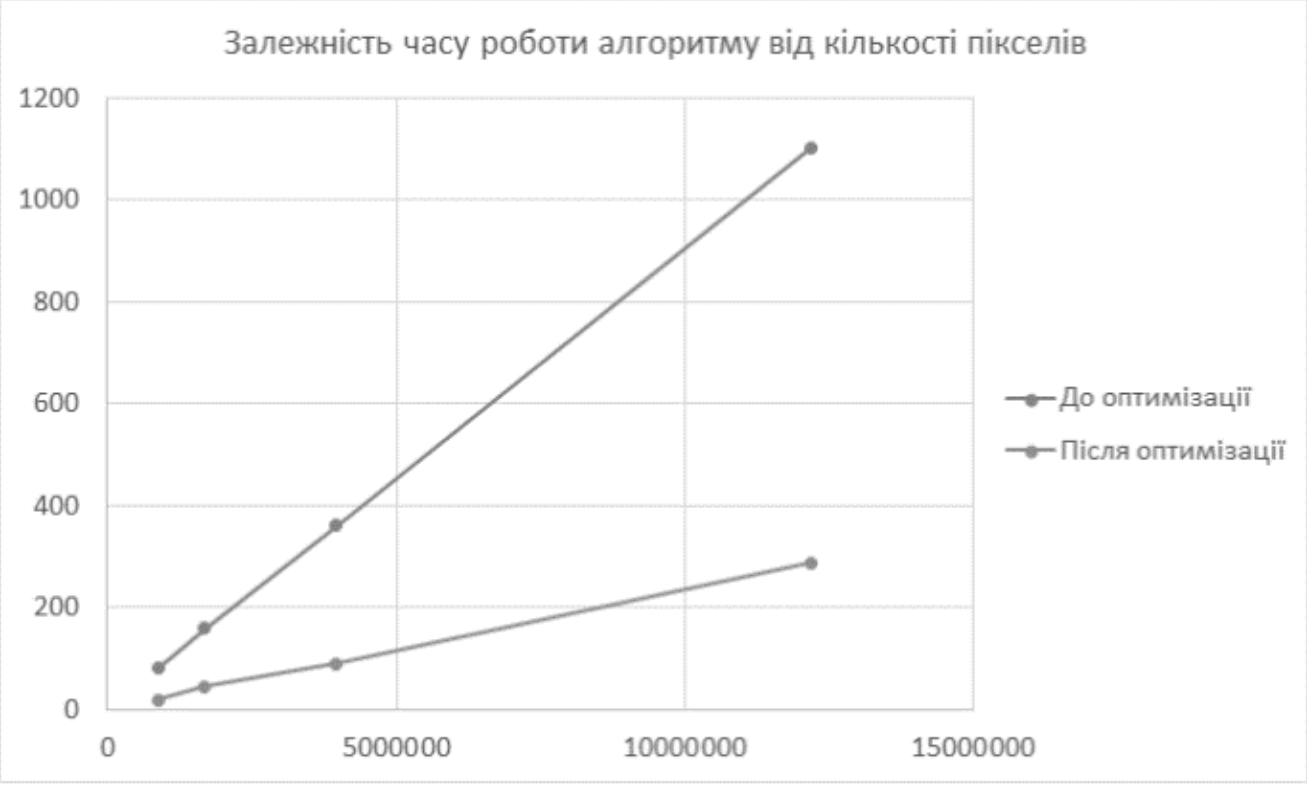
ДП.454400-06-99
Система моніторингу використання зображень у соціальній мережі Reddit. Структура класів алгоритму виділення метаданих зображення. Діаграма класів



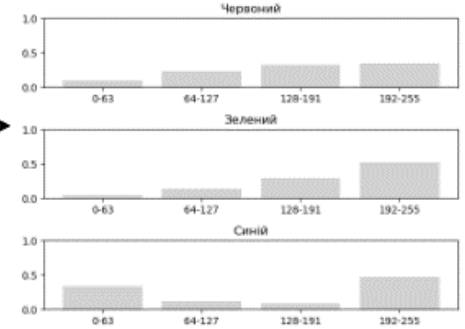
ДП.454400-07-99
Система моніторингу використання зображень у соціальній мережі Reddit.
Розгортання системи моніторингу.
Діаграма розгортання

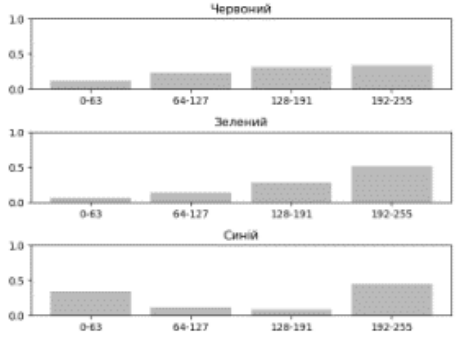
Результат оптимізації функції обробки зображень

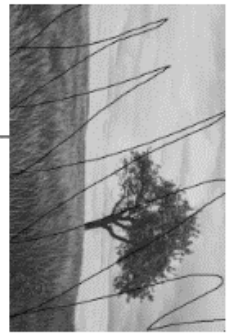
Роздільна здатність	Розмір, КБ	До оптимізації, мс	Після оптимізації, мс
800x1086	101	80	18
1125x1500	74	158	44
2060x1920	765	361	90
4032x3024	2681	1104	287

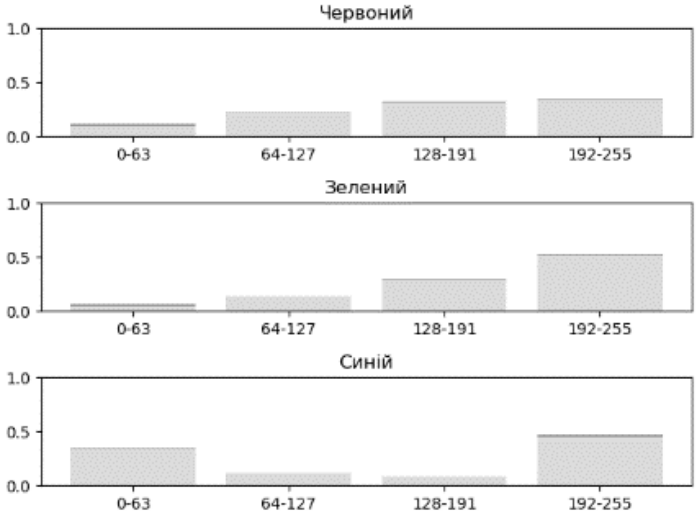


Перетворення зображення в гістограми та порівняння









Додаток 2

Лістинги програм

Компонента списку зображень

```
<template>
    <div>
        <div class="uk-text-center">
            <h1 class="uk-heading-primary">
                {{title}}
            </h1>
        </div>
        <vk-grid matched>
            <transition-group name="image-list" tag="div">
                <template v-for="item in images">
                    <top-image v-if="!selectedImage || selectedImage.url
!== item.url" :key="item.url + item.count" :image="item" @opened="(i) =>
selectImage(i)"> </top-image>
                    <selected-image v-else :image="item" :key="item.url +
item.count" @closed="() => selectImage(null)">dfdsf</selected-image>
                </template>
            </transition-group>
        </vk-grid>
    </div>
</template>

<script>
import TopImage from './../components/TopImage.vue'
import SelectedImage from './../components/SelectedImage.vue'
export default {
    name: 'image-list',
    props: ['images', 'title'],
    components: {TopImage, SelectedImage},
    data: function() {
        return {
            selectedImage: null
        }
    },
    methods: {
        selectImage(i) {
            this.selectedImage = i;
        }
    }
}
</script>
```

```
<style>
.image-list-enter-active, .image-list-leave-active {
  transition: all 3s;
}
.image-list-enter, .image-list-leave-to /* .list-leave-active below version
2.1.8 */ {
  opacity: 0;
  transform: translateY(30px);
}

.image-list-move {
  transition: transform 1s;
}

.image-list-leave-active {
  position: absolute;
}
</style>
```

Компонента обраного зображення

```
<template>
    <div class="uk-width-1-1 uk-margin uk-section uk-section-secondary uk-
padding">
        <div class="uk-container">
        <vk-grid matched>
            <div class="uk-width-1-3">
                <span class="selected-image-container">
                    <img :src="image.url" class="selected-image">
                </span>
                <div class="uk-panel uk-panel-scrollable commentsscroll">
                    <div v-for="comment in image.comments" :key="comment">
                        <vk-button-link :class="{'uk-button-primary':
selectedComment == comment}" @click="() => selectComment(comment)">
                            {{getPublicationName(comment)}}
                        </vk-button-link>
                    </div>
                </div>
            </div>
            <div class="uk-width-2-3">
                <div class="container redditpost">
                    <h2>{{selectedCommentTitle}}</h2>
```

```
                    <p>{{selectedCommentText}}</p>

                    <vk-button-link target='_blank'
:href="selectedComment">{{selectedComment}}</vk-button-link>
                </div>
            </div>
        </vk-grid>
        </div>
    </div>
</template>

<script>
import axios from 'axios';

export default {
    name: 'top-image',
    props: ['image'],
    data: function() {
        return {
            selectedComment: null,
            selectedCommentTitle: null,
            selectedCommentText: null
        }
    },
    watch: {
        image: {
            immediate: true,
            handler: function (cur) {
                if(!cur)
                    return;

                setTimeout(() => this.selectComment(cur.comments[0]),
1000);
            }
        }
    },
    methods: {
        selectComment(c) {
            if(this.selectedComment == c)
                return;

            this.selectedComment = c;
```

```
            axios.post('/api/redditdata/findpost', {url: c})
                .then(r => {
                    this.selectedCommentTitle = r.data.title;
                    this.selectedCommentText = r.data.text;
                });
        },
        getPublicationName(c) {
            var split = c.split('/').filter(s => s.length !== 0);
            var underscoreName = split[split.length - 1];
            var underscoreSplit = underscoreName.split('_');

            var name = underscoreSplit.join(' ');
            return name + " in " + split[3];
        }
    }
}
</script>

<style lang="scss">

.selected-image-container {
    text-align: center;
}

.selected-image {
    border: 1px white;
    max-height: 400px;
    object-fit: contain;
}

.uk-button {
    word-wrap: break-word;
}

div.commentsscroll {
    max-height: 300px;
    white-space: normal;

    & .uk-button {
        white-space: normal;
    }
}
```

```
.redditpost {
    display: flex;
    flex-direction: column;
    p {
        flex-grow: 1;
    }
}


</style>
```

Компонента форми входу та реєстрації

```
<template>
    <div>
    <vk-tabs align="center">
        <vk-tabs-item title="Sign In">
            <div class="signin" @keyup="(e) => e.keyCode == 13 &&
signin()">
                <div class="uk-margin uk-text-center">
                    <div class="uk-inline uk-width-1-2">
                        <span class="uk-form-icon"><vk-icons-user></vk-
icons-user></span>
                        <input class="uk-input" type="text" v-
model="signinUsername">
                    </div>
                </div>
                <div class="uk-margin uk-text-center">
                    <div class="uk-inline uk-width-1-2">
                        <span class="uk-form-icon"><vk-icons-lock></vk-
icons-lock></span>
                        <input class="uk-input" type="password" v-
model="signinPassword">
                    </div>
                </div>
                <div class="uk-margin uk-text-center">
                    <vk-button type="primary" class="uk-inline uk-width-1-
2" :disabled="signinDisabled" @click="signin">LOGIN</vk-button>
                </div>
            </div>
        </vk-tabs-item>
        <vk-tabs-item title="Sign Up">
```

```
            <form class="uk-form-horizontal" @keyup="(e) => e.keyCode == 13
&& signup()" v-if="!confirmEmail">

                <div class="uk-margin">
                    <label class="uk-form-label"
for="username">Username</label>
                    <div class="uk-form-controls">
                        <input class="uk-input" id="username" type="text"
placeholder="Username" v-model="signupUsername">
                    </div>
                </div>

                <div class="uk-margin">
                    <label class="uk-form-label" for="email">Email
address</label>
                    <div class="uk-form-controls">
                        <input class="uk-input" id="email" type="text"
placeholder="you@company.com" v-model="signupEmail">
                    </div>
                </div>


                <div class="uk-margin">
                    <label class="uk-form-label" for="pwd">Password</label>
                    <div class="uk-form-controls">
                        <input class="uk-input" id="pwd" type="password"
placeholder="Password" v-model="signupPassword">
                    </div>
                </div>

                <div class="uk-margin">
                    <label class="uk-form-label" for="confirm">Password
confirmation</label>
                    <div class="uk-form-controls">
                        <input class="uk-input" id="confirm"
type="password" placeholder="Password" v-model="signupConfirmPassword">
                    </div>
                </div>

                <div class="uk-margin uk-text-center">
                    <vk-button type="primary" class="uk-inline uk-width-1-
2" :disabled="signupDisabled" @click="signup">SUBMIT</vk-button>
                </div>
```

```
            </form>
            <span v-if="confirmEmail">
                Please confirm your email {{signupEmail}} via clicking the
link in confirmation email <a @click="confirmEmail = false"> or try again
</a>
            </span>
        </vk-tabs-item>
  </vk-tabs>
  </div>
</template>

<script>
import Axios from 'axios';

export default {
    data: function () {
        return {
            signinUsername: "",
            signinPassword: "",
            signupUsername: "",
            signupPassword: "",
            signupConfirmPassword: "",
            signupEmail: "",
            confirmEmail: false
        }
    },
    methods: {
        signin() {
            if (this.signinDisabled)
                return;

            Axios.post('/api/users/signin', {username: this.signinUsername,
password: this.signinPassword})
                .then((r) => {
                    this.$auth.signin(r.data.token);
                })
                .catch(error => {
                    this.$notifications.error(error.response.data);
                });
        },
        signup() {
            if (this.signupDisabled)
```

```
                return;

            Axios.post('/api/users/signup', {username: this.signupUsername,
password: this.signupPassword, confirmPassword: this.signupConfirmPassword,
email: this.signupEmail})
                .then(() => {
                    this.$notifications.success('Registration successful!
Please confirm your account by clicking on the link in confirmation
email');
                    this.confirmEmail = true;
                })
                .catch(error => {
                    this.$notifications.error(error.response.data);
                });
        }
    },
    computed: {
        signinDisabled() {
            var validUsername = this.signinUsername &&
this.signinUsername.length;
            var validPassword = this.signinPassword &&
this.signinPassword.length;
            return !(validUsername && validPassword);
        },
        signupDisabled() {
            var validUsername = this.signupUsername &&
this.signupUsername.length;
            var validPassword = this.signupPassword &&
this.signupPassword.length;
            var validConfirmPassword = this.signupConfirmPassword &&
this.signupConfirmPassword.length;
            var validEmail = this.signupPassword &&
this.signupPassword.length;
            return !(validUsername && validPassword && validConfirmPassword
&& validEmail);
        }
    }
}
</script>

<style>

</style>
```

Компонента вибору фільтрування сабредитів

```
<template>
    <div>
        <h1 class="uk-heading-line uk-text-
center"><span>Subreddits</span></h1>
        <p class="uk-text-lead">By default, all are selected</p>

        <div class="uk-margin">
            <input class="uk-input" @input="(e) => changeFilterDebounce(e)"
placeholder="Find subreddit">
        </div>

        <DynamicScroller
            class="scroller"
            :items="filteredSubreddits"
            :min-item-size="32"
            key-field="name">
            <template v-slot="{ item, index, active }">
                <DynamicScrollerItem
                    :item="item"
                    :active="active"
                    :size-dependencies="[
                        item.name,
                    ]"
                    :data-index="index">
                    <label><input class="uk-checkbox" type="checkbox"
@change="() => changeCheckbox()" v-model="item.value"> {{item.name}}
</label>
                </DynamicScrollerItem>
            </template>
        </DynamicScroller>

        <div class="uk-margin">
            <vk-button class="uk-margin-right" @click="() =>
selectAll()">Select All</vk-button>

            <vk-button @click="() => clearAll()">Clear All</vk-button>
        </div>
    </div>
```

```
</template>

<script>
import _ from 'lodash';

export default {
    props: ['allSubreddits'],
    data() {
        return {
            subredditFilter: "",
            subreddits: []
        }
    },
    watch: {
        allSubreddits(newValue, old) {
            if(!newValue || newValue == old)
                return;

            var oldValuesMap = {};

            for(const oldValue of this.subreddits) {
                oldValuesMap[oldValue.name] = oldValue.value;
            }

            this.subreddits = newValue
                .sort((a, b) =>
a.toLowerCase().localeCompare(b.toLowerCase()))
                .map(s => ({name: s, value: oldValuesMap[s]}));
        }
    },
    methods: {
        changeCheckbox() {
            this.$subredditStore.setSubreddits(this.subreddits.filter(s =>
s.value).map(s => s.name));
        },
        clearAll() {
            for(var s of this.filteredSubreddits) {
                s.value = false;
            }
            this.$subredditStore.setSubreddits(this.subreddits.filter(s =>
s.value).map(s => s.name));
        },
        selectAll() {
```

```
            for(var s of this.filteredSubreddits) {
                s.value = true;
            }
            this.$subredditStore.setSubreddits(this.subreddits.filter(s =>
s.value).map(s => s.name));
        },
        changeFilterDebounce:  _.debounce(function(e) {
this.changeFilter(e)}, 200),
        changeFilter(e) {
            this.subredditFilter = e.target.value;
        }
    },
    computed: {
        filteredSubreddits() {
            if(this.subredditFilter && this.subredditFilter.length) {
                var filter = this.subredditFilter.toLowerCase();
                return this.subreddits.filter(s =>
s.name.toLowerCase().includes(filter));
            }

            return this.subreddits;
        }
    }

}
</script>

<style>
.scroller {
  height: 400px;
  overflow-y: auto;
}

.scroller > label {
  height: 32%;
  padding: 0 12px;
  display: flex;
  align-items: center;
}
</style>
```

Компонента відображення зображення

```
<template>
    <div class="uk-card uk-card-default uk-card-hover uk-width-1-6 uk-
margin">
        <div class="uk-card-header">
            <img :src="image.url">
        </div>
        <div class="uk-card-body">
            <div class="uk-text-lead uk-margin-bottom">
                Count: {{image.count}}
            </div>
        </div>
        <div class="uk-card-footer">
            <vk-button-link  @click="() => this.$emit('opened', image)">
                Posts +
            </vk-button-link>
        </div>
    </div>
</template>

<script>
export default {
    name: 'top-image',
    props: ['image']
}
</script>

<style lang="css" scoped>
    a {
        word-wrap: break-word;
    }
</style>
```

Контроллер реєстрації та автентифікації

```
using Amazon.CognitoIdentityProvider;
using Amazon.CognitoIdentityProvider.Model;
using KPI.RedditMonitor.Api.Config;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
```

```
using Microsoft.Extensions.Options;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace KPI.RedditMonitor.Api.Controllers
{
    [ApiController]
    [AllowAnonymous]
    [Route("api/[controller]")]
    public class UsersController : ControllerBase
    {
        private readonly IAmazonCognitoIdentityProvider _cognito;
        private readonly IOptions<CognitoOptions> _options;
        private readonly ILogger<UsersController> _log;

        public UsersController(IAmazonCognitoIdentityProvider cognito,
IOptions<CognitoOptions> options, ILogger<UsersController> log)
        {
            _cognito = cognito;
            _options = options;
            _log = log;
        }

        [HttpPost("signin")]
        public async Task<ActionResult<LoginResponse>> Signin(LoginRequest
request)
        {
            var auth = new AdminInitiateAuthRequest
            {
                ClientId = _options.Value.ClientId,
                UserPoolId = _options.Value.UserPoolId,
                AuthFlow = AuthFlowType.ADMIN_NO_SRP_AUTH
            };

            auth.AuthParameters.Add("USERNAME", request.Username);
            auth.AuthParameters.Add("PASSWORD", request.Password);

            try
            {
```

```
                var authResponse = await
_cognito.AdminInitiateAuthAsync(auth);

                if (authResponse.AuthenticationResult == null)
                {
                    _log.LogError("Got response @{response}, error",
authResponse);
                    return BadRequest();
                }

                return new LoginResponse
                {
                    Token = authResponse.AuthenticationResult.AccessToken,
                    ExpiresAt =
DateTime.UtcNow.AddSeconds(authResponse.AuthenticationResult.ExpiresIn)
                };
            }
            catch(UserNotFoundException e)
            {
                _log.LogInformation(e, $"User #${request.Username} does not
exist");
                return BadRequest("Incorrect username or password");
            }
            catch(NotAuthorizedException e)
            {
                _log.LogInformation(e, $"User attempted to log in with
incorrect password for username #{request.Username}");
                return BadRequest("Incorrect username or password");
            }
            catch(UserNotConfirmedException e)
            {
                _log.LogInformation(e, $"User #{request.Username} is not
confirmed yet");
                return BadRequest("Please confirm your account (or contact
us at webmaster@reddit.knine.xyz)");
            }
        }

        [HttpPost("signup")]
        public async Task<IActionResult> Signup(SignupRequest request)
        {
            if (!request.Password.Equals(request.ConfirmPassword))
```

```
                return BadRequest("Password and confirmation should be
equal");

            var auth = new SignUpRequest
            {
                ClientId = _options.Value.ClientId,
                Password = request.Password,
                Username = request.Username
            };

            var emailAttribute = new AttributeType
            {
                Name = "email",
                Value = request.Email
            };
            auth.UserAttributes.Add(emailAttribute);

            try
            {
                var authResponse = await _cognito.SignUpAsync(auth);

                if (authResponse.HttpStatusCode !=
System.Net.HttpStatusCode.OK)
                {
                    _log.LogError("Got response @{response}, error",
authResponse);
                    return BadRequest();
                }

                return Ok();
            }
            catch(InvalidPasswordException e)
            {
                _log.LogWarning(e, "User attempted to create an account
with invalid password");
                return BadRequest("Please choose a stronger password: with
uppercase and lowercase letters, numbers");
            }
            catch(UsernameExistsException e)
            {
                _log.LogWarning(e, "User tried to create an account with
duplicate username");
                return BadRequest("Please choose another username");
```

```
            }

        }
    }

    public class LoginRequest
    {
        [Required]
        public string Username { get; set; }
        [Required]
        public string Password { get; set; }
    }

    public class LoginResponse
    {
        public string Token { get; set; }
        public DateTime ExpiresAt { get; set; }
    }

    public class SignupRequest
    {
        [Required]
        public string Username { get; set; }
        [Required]
        public string Email { get; set; }
        [Required]
        public string Password { get; set; }
        [Required]
        public string ConfirmPassword { get; set; }
    }
}
```

Створювач метаданих зображення

```
using System.IO;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.Advanced;

namespace KPI.RedditMonitor.Application.Similarity
{
    public static class ImageFeatureFactory
    {
```

```
        public static ImageFeatures Create(Stream content)
        {
            var features = new ImageFeatures();
            using (var image = Image.Load(content))
            {
                const int buckets = 4;
                const int minRgb = 0;
                const int maxRgb = 255;

                var red = features.AddHistogram("red", buckets, minRgb,
maxRgb);
                var green = features.AddHistogram("green", buckets, minRgb,
maxRgb);
                var blue = features.AddHistogram("blue", buckets, minRgb,
maxRgb);

                foreach (var pixel in image.GetPixelSpan())
                {
                    red.Add(pixel.R);
                    green.Add(pixel.G);
                    blue.Add(pixel.B);
                }
            }

            return features;
        }
    }
}
```

Клас з метаданими зображення

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Serialization;

namespace KPI.RedditMonitor.Application.Similarity
{
    /// <summary>
    /// Separates all pixels into buckets by their value
    /// Image Similarity preprocessing via feature histogram method
    /// </summary>
```

```
    public class ImageFeatures
    {
        private readonly Dictionary<string, Histogram> _histograms;

        public ImageFeatures()
        {
            _histograms = new Dictionary<string, Histogram>();
        }

        public Histogram AddHistogram(string name, int buckets, double
minValue, double maxValue)
        {
            var histogram = new Histogram(name, buckets, minValue,
maxValue);
            _histograms.Add(name, histogram);
            return histogram;
        }

        public Dictionary<string, double[]> GetNormalizedBuckets()
        {
            return _histograms.ToDictionary((kv) => kv.Key, kv =>
kv.Value.GetNormalizedBuckets());
        }
    }
}
```

Клас-функція обробки зображень

```
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using KPI.RedditMonitor.Application.Similarity;
using KPI.RedditMonitor.Data;
using Microsoft.Extensions.Configuration;
using MongoDB.Driver;
using Newtonsoft.Json;
using SixLabors.ImageSharp;
using SixLabors.Memory;
using Microsoft.Extensions.Logging.Abstractions;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be
converted into a .NET class.
[assembly:
LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace KPI.RedditMonitor.ImageProcessing
{
    public class Function
    {
        private readonly ImagePostsRepository _repository;
        private readonly HttpClient _httpClient;

        /// <summary>
        /// Default constructor. This constructor is used by Lambda to
construct the instance. When invoked in a Lambda environment
        /// the AWS credentials will come from the IAM role associated with
the function and the AWS region will be set to the
        /// region the Lambda function is executed in.
        /// </summary>
        public Function()
        {
            var config = new ConfigurationBuilder()
                .AddJsonFile($"appsettings.Development.json", true)
                .AddEnvironmentVariables()
                .Build();
            var mongoClient = new
MongoClient(config["MongoDb:ConnectionString"]);

            _repository = new ImagePostsRepository(mongoClient,
NullLogger<ImagePostsRepository>.Instance);
            _httpClient = new HttpClient();
            // Configuration.Default.MemoryAllocator =
ArrayPoolMemoryAllocator.CreateWithModeratePooling();
        }


        /// <summary>
        /// This method is called for every Lambda invocation. This method
takes in an SQS event object and can be used
        /// to respond to SQS messages.
        /// </summary>
        /// <param name="evnt"></param>
```

```
        /// <param name="context"></param>
        /// <returns></returns>
        public async Task FunctionHandler(SQSEvent evnt, ILambdaContext
context)
        {
            var maxImageSize = 3 * 1024 * 1024;
            var inserted = 0;

            foreach (var record in evnt.Records)
            {
                var imagePost =
JsonConvert.DeserializeObject<ImagePost>(record.Body);
                using (var response = await
_httpClient.GetAsync(imagePost.ImageUrl,
HttpCompletionOption.ResponseHeadersRead))
                {
                    if(!response.IsSuccessStatusCode)
                    {
                        context.Logger.LogLine($"StatusCode
{response.StatusCode} was not successful while doing request to
{imagePost.ImageUrl}, so skipping image");
                        continue;
                    }

                    if (response.Content.Headers.TryGetValues("Content-
Length", out var values)
                        && values.Any((v) => int.Parse(v) > maxImageSize))
                    {
                        context.Logger.LogLine($"Could not save image
{imagePost.ImageUrl} - it was too big");
                        continue;
                    }

                    using (var content = await
response.Content.ReadAsStreamAsync())
                    {
                        var features = ImageFeatureFactory.Create(content);

                        imagePost.FeatureBuckets =
features.GetNormalizedBuckets();
                    }
                }
```

```
                var duplicateError = "E11000 duplicate key error index";
                try
                {
                    await _repository.Add(imagePost);
                    inserted++;
                }
                catch (MongoWriteException e)
                    when (e.Message.Contains(duplicateError))
                {
                    context.Logger.LogLine($"Could not save
{imagePost.ImageUrl}, seems that it was already inserted");
                }

            }

            context.Logger.LogLine($"Inserted {inserted} posts");
        }
    }
}
```

Клас з реалізованим алгоритмом пошуку по базі MongoDB

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using MongoDB.Bson;
using MongoDB.Bson.Serialization;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Driver;

namespace KPI.RedditMonitor.Data
{
    public class ImagePostsRepository
    {
        private readonly ILogger<ImagePostsRepository> _log;
        private readonly IMongoClient _client;

        public ImagePostsRepository(IMongoClient client,
ILogger<ImagePostsRepository> log)
        {
```

```
            _log = log;
            _client = client;
        }

        private IMongoCollection<ImagePost> GetCollection()
        {
            return _client.GetDatabase("reddit")
                .GetCollection<ImagePost>("posts");
        }

        public async Task AddRange(IEnumerable<ImagePost> posts)
        {
            await GetCollection().InsertManyAsync(posts);
        }

        public async Task<List<string>> GetAllSubreddits()
        {
            var r = await GetCollection().DistinctAsync(a => a.Subreddit,
(a) => a.Ignore != true);

            var results = new List<string>();
            while(await r.MoveNextAsync())
            {
                results.AddRange(r.Current);
            }

            return results;
        }

        public async Task Add(ImagePost imagePost)
        {
            await GetCollection().InsertOneAsync(imagePost);
        }

        /// <summary>
        /// Algorithm uses Intersection to compare color histograms
        /// </summary>
        /// <param name="features"></param>
        /// <param name="top"></param>
        /// <returns></returns>
        public async Task<List<TopImage>> FindSimilar(Dictionary<string,
double[]> features, int top = 10, string[] subreddits = null)
        {
```

```
            var featuresArray =
features["red"].Concat(features["green"]).Concat(features["blue"]);
            var featuresBson = BsonArray.Create(featuresArray);

            var bsonSubreddits = new BsonArray();

            foreach(var a in subreddits ?? new string[0]) {
                bsonSubreddits.Add(a);
            }

            var query = @"
[{
        $match: {
            Ignore: {
                '$in': [null, false]
            }" +
            (subreddits?.Length > 0 ? @",
            Subreddit: {
                '$in': " + bsonSubreddits.ToJson() + @"
            }" : string.Empty) +
@"      }
    },
    {
        $addFields: {
            allFeatures: {
                $zip: {
                    inputs: [" + featuresBson + @", {$concatArrays:
['$FeatureBuckets.red', '$FeatureBuckets.green', '$FeatureBuckets.blue']}]
                }
            }
        }
    },
    {
        $unwind: {
            path: '$allFeatures'
        }
    },
    {
        $group: {
            _id: '$_id',
            intersection: {
                $sum: {
                    $min: '$allFeatures'
```

```
                }
            },
            image: {
                '$first': '$$ROOT'
            }
        }
    },
    {
        $group: {
            _id: '$image.ImageUrl',
            image: {
                '$first': '$image'
            },
            comments: {
                '$addToSet': '$image.Url'
            },
            intersection: {
                '$first': '$intersection'
            }
        }
    },
    {
        $sort: {
            'intersection': -1
        }
    },
    {
        $limit: 5
    }
]
";

            var pipelines =
BsonSerializer.Deserialize<BsonDocument[]>(query).ToList();

            using (var result = await
GetCollection().AggregateAsync<AggregateResult>(pipelines, new
AggregateOptions()
            {
                AllowDiskUse = true
            }))
            {
                await result.MoveNextAsync();
```

```
                return result.Current.Select(c => new TopImage
                {
                    Comments = c.comments,
                    Count = c.comments.Length,
                    Url = c.image.ImageUrl
                }).ToList();
            }
        }

        private class AggregateResult
        {
            public string Id { get; set; }

            public double intersection { get; set; }

            public ImagePost image { get; set; }

            public string[] comments { get; set; }
        }

    }
}
```

Клас, що реалізує пошук публікацій з однаковими посиланнями на зображення

```
using Microsoft.Extensions.Logging;
using MongoDB.Bson;
using MongoDB.Driver;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace KPI.RedditMonitor.Data
{
    public class TopImageDbAdapter
    {
        private readonly ILogger<TopImageDbAdapter> _log;
        private readonly IMongoCollection<ImagePost> _collection;
```

```
        public TopImageDbAdapter(IMongoClient client,
ILogger<TopImageDbAdapter> log)
        {
            _log = log;
            _collection =
client.GetDatabase("reddit").GetCollection<ImagePost>("posts");
        }

        public Task Ignore(string imageUrl, bool ignoreValue)
        {
            if (imageUrl.StartsWith("https://"))
                imageUrl = imageUrl.Substring("https://".Length);

            var update = Builders<ImagePost>.Update.Set(i => i.Ignore,
ignoreValue);
            return _collection.UpdateManyAsync(i =>
i.ImageUrl.Contains(imageUrl), update);
        }

        public async Task<List<TopImage>> GetTop(int count, bool
showIgnored, string[] subreddits)
        {
            var query = _collection.Aggregate();

            if (!showIgnored)
                query = query.Match(image => image.Ignore != true);
            else
                query = query.Match(image => image.Ignore == true);

            if(subreddits?.Length > 0)
            {
                var filter = Builders<ImagePost>.Filter.In(a =>
a.Subreddit, subreddits);
                query = query.Match(filter);
            }

            _log.LogInformation(query.ToString());

            var images = await query
                .Group(image => image.ImageUrl, group => new TopImage
                {
                    Count = group.Count(),
                    Url = group.Key,
```

```
                    Comments = group.Select(g => g.Url)
                })
                .Sort(new BsonDocument() { { "Count", -1 } })
                .Limit(count)
                .ToListAsync();

            foreach (var topImageDto in images)
            {
                topImageDto.EnsureUrlFormat();
            }

            // Group again gue to adjustments
            return images.GroupBy(i => i.Url).Select(g => new TopImage
            {
                Comments = g.SelectMany(gg => gg.Comments).ToArray(),
                Count = g.Sum(gg => gg.Count),
                Url = g.Key
            }).ToList();
        }

        public async Task<TopImageCount> GetCount()
        {
            var counts = await _collection.Aggregate().Group((dto) =>
dto.Ignore, g => new { g.Key, Count = g.Count() }).ToListAsync();
            return new TopImageCount()
            {
                All = counts.Select(c => c.Count).Sum(),
                Ignored = counts.Where(c => c.Key == true).Sum(c =>
c.Count)
            };
        }
    }

    public class TopImage
    {
        public void EnsureUrlFormat()
        {
            if (!Url.StartsWith("http"))
                Url = $"https://{Url}";

            Comments = Comments.Select(c =>
$"https://reddit.com{c}").Distinct();
        }
```

```
        public string Url { get; set; }

        public int Count { get; set; }

        public IEnumerable<string> Comments { get; set; }
    }

    public class TopImageCount
    {
        public int All { get; set; }

        public int Ignored { get; set; }
    }
}
```

Клас моніторингу нових публікацій Reddit

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Reddit;
using Reddit.Controllers.EventArgs;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace KPI.RedditMonitor.Collector.RedditPull.Collectors
{
    public class RedditNetCollector : IRedditCollector
    {
        private readonly ILogger<RedditNetCollector> _log;
        private readonly RedditOptions _options;

        public RedditNetCollector(IOptions<RedditOptions> options,
ILogger<RedditNetCollector> log)
        {
            _options = options.Value;
            _log = log;
        }
        public Task SubscribeOnEntries(Action<RedditPost> callback,
CancellationToken cancellationToken)
        {
```

```
            var api = new RedditAPI(_options.ClientId,
_options.RefreshToken, _options.ClientSecret);

            var all = api.Subreddit("all");
            EventHandler<PostsUpdateEventArgs> newUpdatedHandler = (s,
args) =>
            {
                foreach(var added in args.Added)
                {
                    var t = added.Listing;
                    callback(new RedditPost(t.Id, t.Title + " " +
t.SelfText + " " + t.URL,
                        t.Permalink.ToString(), t.CreatedUTC, added.NSFW,
added.Subreddit));
                }
            };
            all.Posts.NewUpdated += newUpdatedHandler;
            all.Posts.MonitorNew();

            var taskCompletionSource = new TaskCompletionSource<bool>();
            cancellationToken.Register(() =>
            {
                all.Posts.MonitorNew();
                all.Posts.NewUpdated -= newUpdatedHandler;
                taskCompletionSource.TrySetResult(true);
            });
            return taskCompletionSource.Task;
        }
    }
}
```

Додаток З

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Система моніторингу використання зображень у соціальній мережі Reddit

Виконав: Шароварський Костянтин Вікторович

Науковий керівник: доцент кафедри ПЗКС, к.т.н., доцент, Сулема Є.С.

Київ – 2019

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити систему моніторингу зображень у соціальній мережі Reddit.

Завдання:

1. Проаналізувати алгоритми аналізу зображень;
2. Розробити систему виділення зображень із публікацій в реальному часі із подальшим аналізом;
3. Протестувати систему на реальних публікаціях в соціальній мережі Reddit.

СОЦІАЛЬНА МЕРЕЖА Reddit

r/dataisbeautiful

Search r/dataisbeautiful

LOG IN SIGN UP

Posted by u/AutoModerator 8 days ago

11

Discussion [Topic][Open] Open Discussion Monday — Anybody can post a general visualization question or start a fresh discussion!

Anybody can post a Dataviz-related question or discussion in the biweekly topical threads. (**Meta is fine too,** but if you want a more direct line to the mods, click here.) If you have a general question you need answered, or a discussion you'd like to start, feel free to make a top-level comment!

Beginners are encouraged to ask basic questions, so please be patient responding to people who might not know as much as yourself.

To view all Open Discussion threads, click here. To view all topical threads, click here.

Want to suggest a biweekly topic? Click here.

21 Comments Share Save ...

Posted by u/Harshaavardhan OC: 1 8 hours ago

6.7k

oc [OC] Where the wealth is concentrated in the world.

ADVERTISEMENT

Ad closed by Google

Stop seeing this ad

Why this ad? ⓘ

BEST OF DATAISBEAUTIFUL

VIEW THIS WEEK'S TOP OC

POSTING RULES

1. A post must be a data visualization.
2. Directly link to the original source article of the visualization (not an image file) or tag the post as [OC] if you made the visualization.

АКТУАЛЬНІСТЬ

- Щогодини створюється понад 2000 публікацій, не враховуючи коментарів;
- Людині без технічних засобів таку кількість інформації проаналізувати неможливо;
- У соціальній мережі Reddit немає вбудованих інструментів аналізу зображень (наприклад, пошуку).

Отже, даний проект спрощує аналіз величезного об’єму даних зображень в реальному часі.

ІНСТРУМЕНТИ РОЗРОБКИ: КЛІЄНТСЬКА ЧАСТИНА

Обрано веб застосунок через простоту встановлення.

HTML, JavaScript, CSS – невід’ємні складові веб розроблення.

VueJS – один із найкращих фреймворків для швидкості розроблення.

ІНСТРУМЕНТИ РОЗРОБКИ: СЕРВЕРНА ЧАСТИНА

C# та .NET Core – кросплатформенні інструменти розроблення, які підтримуються Microsoft.

Вихідний код фреймворку .NET Core є у вільному доступі.

Надбудова .NET Core для веб застосунків – ASP.NET Core є однією з найшвидших на ринку веб фреймворків.

ПРОДУКТИВНІСТЬ ASP.NET CORE

7 місце з 307 протестованих за версією TechEmpower.

Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE (307 tests)

Rnk	Framework	Best performance (higher is better)	
1	ulib	7,034,945	100.0%
2	fasthttp	7,033,219	100.0%
3	libreactor	7,027,558	99.9%
4	wizzardo-http	7,026,401	99.9%
5	ulib-plaintext_fit	7,022,947	99.8%
6	actix-raw	7,021,312	99.8%
7	aspcore	7,016,017	99.7%
8	hyper	7,013,819	99.7%
9	tokio-minihttp	7,009,815	99.6%
10	httpbeast	7,008,766	99.6%

ІНСТРУМЕНТИ РОЗРОБКИ: БІБЛІОТЕКА КОМУНІКАЦІЇ ІЗ Reddit

Знайдено дві можливі бібліотеки для комунікації - RedditSharp та Reddit.NET.

RedditSharp мала проблеми із продуктивністю.

Тому було вирішено обрати другу, менш популярну, бібліотеку Reddit.NET.

ІНСТРУМЕНТИ РОЗРОБКИ: ХМАРНИЙ ПРОВАЙДЕР

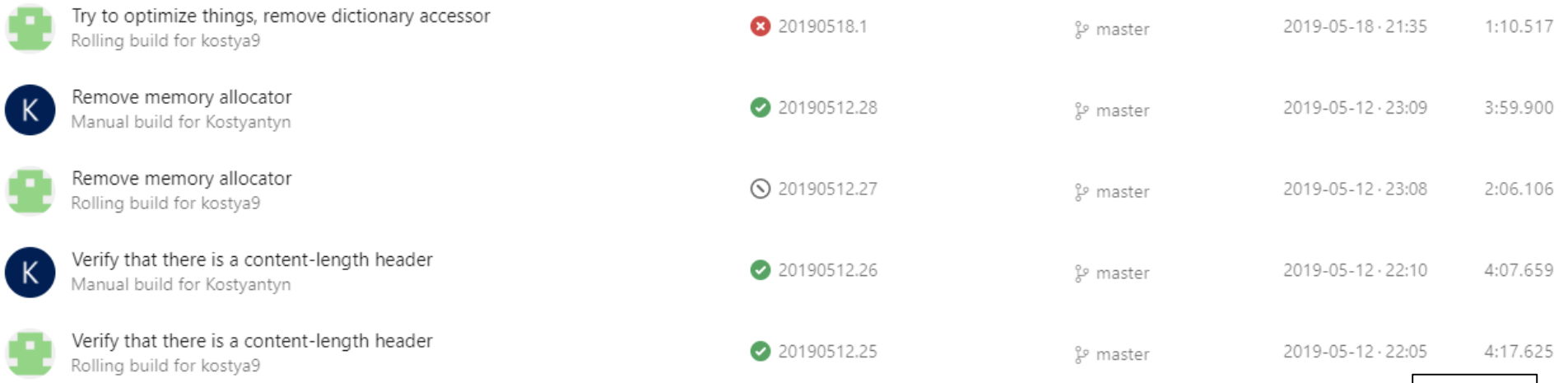
Для забезпечення масштабованості було вирішено використовувати хмарні технології.

Провайдером цих технологій було обрано Amazon AWS через малу ціну та величезний обсяг послуг.

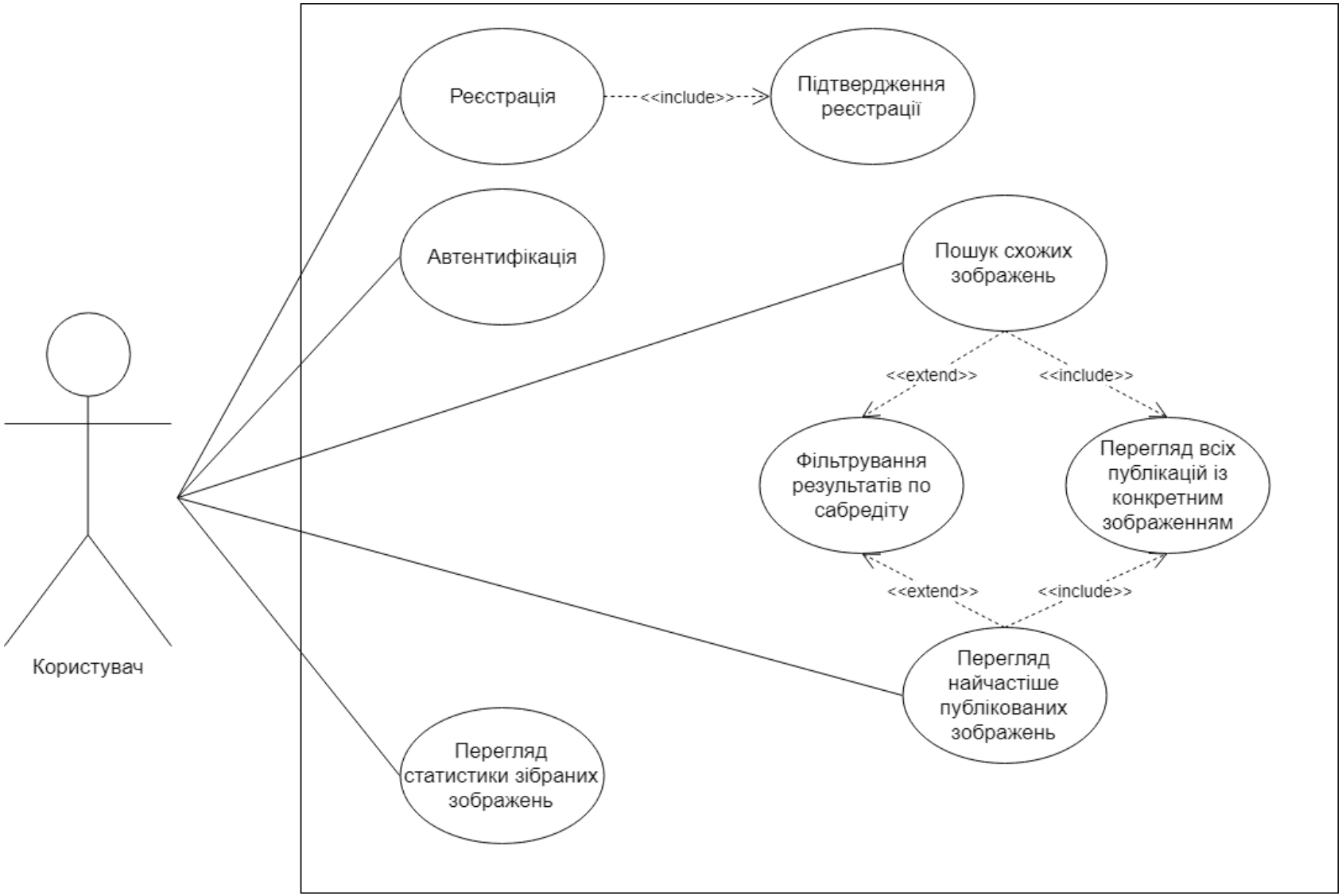
ІНСТРУМЕНТИ РОЗРОБКИ: РОЗГОРТАННЯ

Автоматичне розгортання при будь-якій зміні вихідного коду за допомогою сервісу CI/CD Azure Pipelines.

Ansible – інструмент конфігурації машини розгортання.



ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ СИСТЕМИ



АРХІТЕКТУРА СИСТЕМИ

Система поділяється на дві частини:

1) підсистема відображення та пошуку – відповідає за відображення всієї інформації та алгоритм пошуку;

2) підсистема збору та обробки даних – моніторинг публікацій на предмет зображень та виділення метаданих з цих зображень.

РОЗРОБЛЕНІ АЛГОРИТМИ: ВИДІЛЕННЯ МЕТАДАНИХ ЗОБРАЖЕННЯ

Метод гістограм:

1. Поділити весь інтервал можливих значень червоного, зеленого та синього кольорів на чотири частини;

2. Кожен піксель розмістити в один із інтервалів трьох гістограм кольорів;

3. Вихідні значення – розподіл кольорів у зображенні.

РОЗРОБЛЕНІ АЛГОРИТМИ: ПОШУК СХОЖИХ ЗОБРАЖЕНЬ

1. Виділити метадані зображення користувача;

2. За допомогою методу перетинань знайти перетини всіх інтервалів гістограм зображення користувача;

3. Знайти зображення з найбільшою характеристикою перетину гістограм у базі даних.

ТЕСТУВАННЯ: ПРОДУКТИВНІСТЬ ФУНКЦІЇ ОБРОБКИ ЗОБРАЖЕНЬ

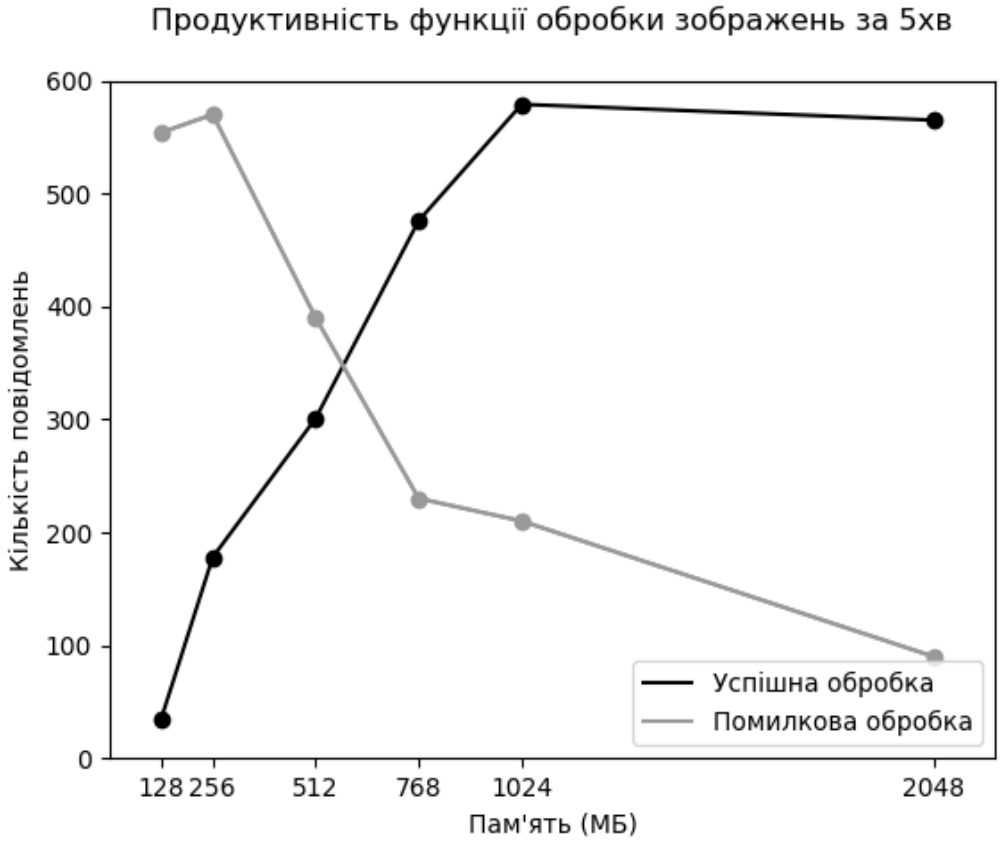
Перші тести продуктивності показали незадовільні результати.

Після певних оптимізацій було досягнута прискорення обробки приблизно в чотири рази.

Приклад однієї з оптимізацій – обрахувати одну із констант перед виконанням алгоритма, замість виконання операції на кожному кроці.

ТЕСТУВАННЯ: ПРОДУКТИВНІСТЬ ФУНКЦІЇ ОБРОБКИ ЗОБРАЖЕНЬ

Після оптимізації, емпіричним шляхом було встановлено, що оптимальна виділена пам’ять для процесу – 1 ГБ.



ТЕСТУВАННЯ: ШВИДКОДІЯ ПОШУКУ

Було проведено тестування на реальних даних, зібраних системою моніторингу публікацій.

Операція пошуку при 20,000 зображеннях в базі даних займає близько 1с.

При збільшенні кількості зображень до 260,000 пошук сповільнюється до 15-20с.

ВИСНОВКИ

1. Розроблено архітектуру ПЗ, що сприяє масштабованості системи при різних навантаженнях.
2. Реалізовано процес збору зображень соціальної мережі Reddit у реальному часі.
3. Реалізовано, протестовано та зроблено оптимізації алгоритму виділення метаданих на основі зібраних зображень.
4. Розроблено алгоритм пошуку схожих зображень за допомогою засобів бази даних MongoDB.

Отже, в результаті роботи над проектом була досягнута мета: розробити систему моніторингу зображень у соціальній мережі Reddit.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

СИСТЕМА МОНІТОРИНГУ ВИКОРИСТАННЯ ЗОБРАЖЕНЬ У СОЦІАЛЬНІЙ МЕРЕЖІ Reddit

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

____ К.В. Шароварський

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Взаємодія користувачі із системою моніторингу використання зображень у соціальній мережі Reddit відбувається за допомогою web-інтерфейсу. Тому об’єктом випробувань є клієнтська частина, побудована на веб технологіях та серверна частина, побудована за допомогою ASP.NET Core, яка надає дані для відображення.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) Працездатність функцій автентифікації та реєстрації
2) Безпека від неавторизованого доступу
3) Поповнення бази даних новими публікаціями
4) Достатня точність пошуку схожих зображень
5) Фільтрація пошуку за сабредітами

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Black Box. Тестується і інтерфейс користувача, і код. Особливість цього методу в тому, що не використовуються знань про внутрішню реалізацію програмного продукту. Для зменшення тестових сценаріїв використовується підхід класів еквівалекції. Тобто, якщо тести виявляють одні і ті самі помилки, потрібно виконувати тільки один із них.

Використовуються наступні методи:

1) Функціональне тестування
2) Тестування продуктивності програмного забезпечення
3) Тестування інтерфейсу

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування інтерфейсу виконується засобами програмного забезпечення Puppeteer, який дозволяє виконувати тести в реальному браузері. Тести продуктивності будуть використовувати бібліотеку BenchmarkDotNet, яка надає характеристики використаної пам’яті та часу роботи процесора.

Працездатність системи перевіряється за допомогою ручного тестування.

Всі поля ручного вводу повинні мати валідуючі правила та відображати інформативне повідомлення у випадку помилки

1) Статичного тестування коду.

2) Тестування web-інтерфейсу в різних браузерах.

3) Тестування при різному навантаженні.

4) Тестування зручності використання.

5) Тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

СИСТЕМА МОНІТОРИНГУ ВИКОРИСТАННЯ ЗОБРАЖЕНЬ У СОЦІАЛЬНІЙ МЕРЕЖІ Reddit

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Є.С. Сулема

Нормоконтроль:

_____ М.В. Онай

Виконавець:

____ К.В. Шароварський

2019

ЗМІСТ

1. Опис структури системи

Система моніторингу з точки зору користувацького інтерфейсу є web-застосунком. Система складається з наступних сторінок:

- Головна сторінка.
- Сторінка автентифікації.
- Сторінка реєстрації.
- Сторінка пошуку схожих зображень.
- Сторінка статистики зображень в базі даних.
- Налаштування фільтрації сабредитів.

Перед тим, як починати використовувати найважливіші функціональні можливості, користувач повинен автентифікуватися. Без виконання цього кроку, система блокує всі свої функції.

Після автентифікації користувач автоматично переходить до сторінки пошуку схожих зображень.

Сторінка пошуку, статистики та налаштувань доступні авторизованому користувачу з будь-якої іншої сторінки за допомогою верхнього меню навігації.

Головну сторінка бачать тільки анонімні користувачі та з неї доступні лише сторінки для реєстрації та автентифікації

2. Процедура автентифікації

При переході на сторінку автентифікації, користувач побачить 2 поля вводу: логін та пароль. Користувач повинен ввести дані в обидва поля для розблокування кнопки входу.

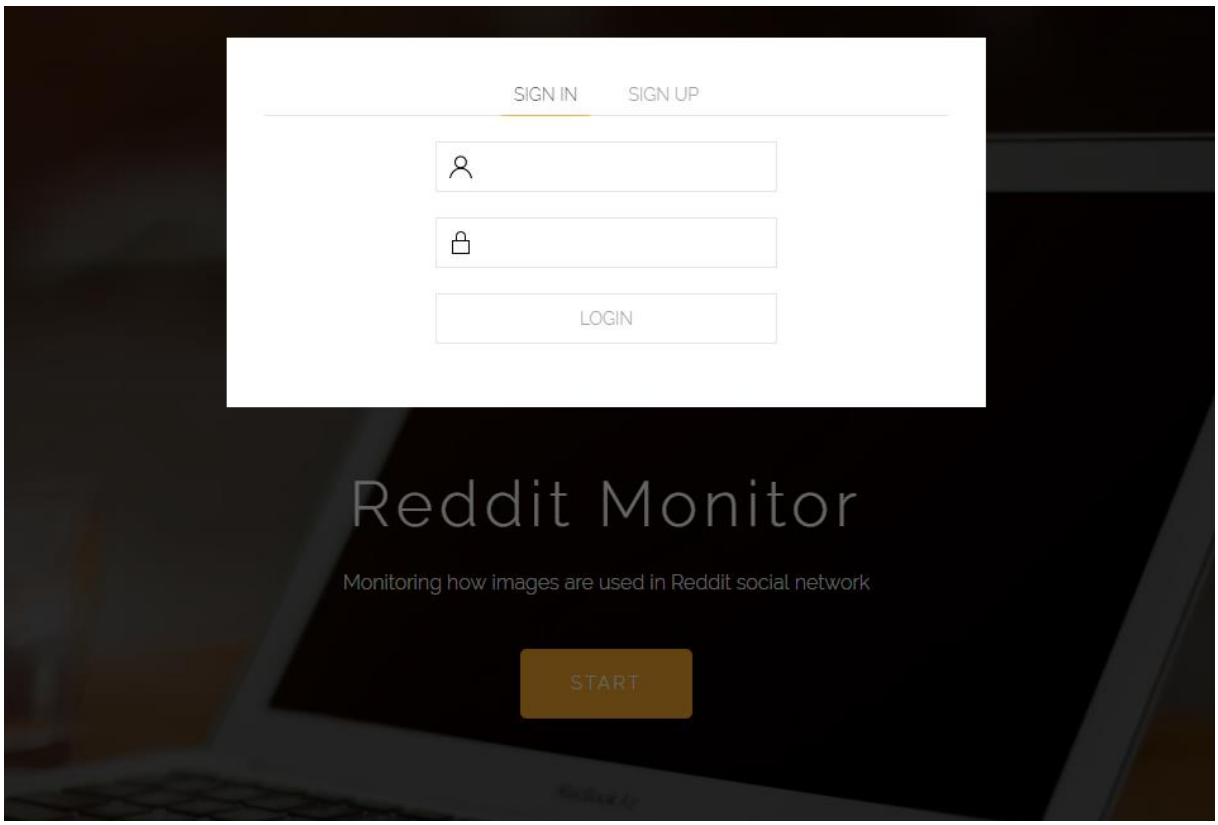


Рис. 1. Сторінка автентифікації

Якщо користувач не підтвердив реєстрацію, то він побачить відповідне повідомлення про помилку.

У випадку неправильного паролю чи неправильному імені користувачі система виводить повідомлення про помилку.

Одразу після автентифікації, користувач перенаправляється на сторінку пошуку схожих зображень

3. Процедура реєстрації

На сторінці реєстрації користувач побачить 4 місця для вводу: ім’я користувача, адреса електронної пошта, пароль та підтвердження паролю.

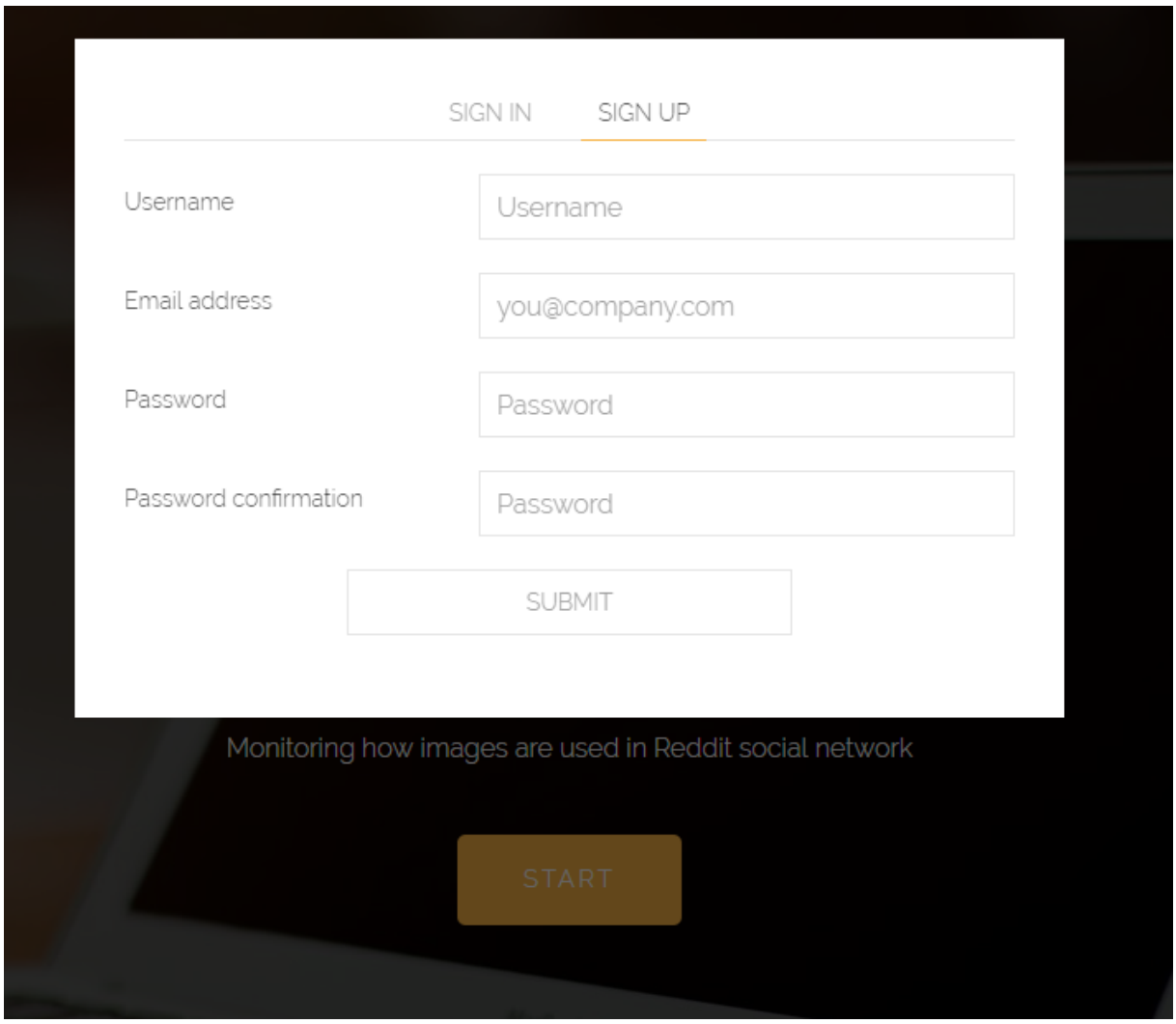


Рис. 2. Сторінка реєстрації

До паролю існують наступні вимоги:

- Мінімум 8 символів.
- Хоч одна цифра.
- Хоч одна буква верхнього регістру.
- Хоч одна буква нижнього регістру.

При порушенні хоч одного пункту, користувачу виводиться тост-повідомлення з повідомленням про недостатню складність паролю.

У випадку, якщо ім’я користувача вже зайняте іншою людиною, система виведе тост-повідомлення з інформацією про це.

Після успішної реєстрації, на пошту користувачу надійшлеться лист із посиланням для підтвердження реєстрації. Тільки після підтвердження реєстрації система дозволить автентифікацію.

4. Пошук схожих зображень

На сторінці пошуку відображається кнопка із повідомленням обрати зображення. Після натискання на неї, за допомогою стандартного інтерфейса браузера можна обрати зображення для пошуку

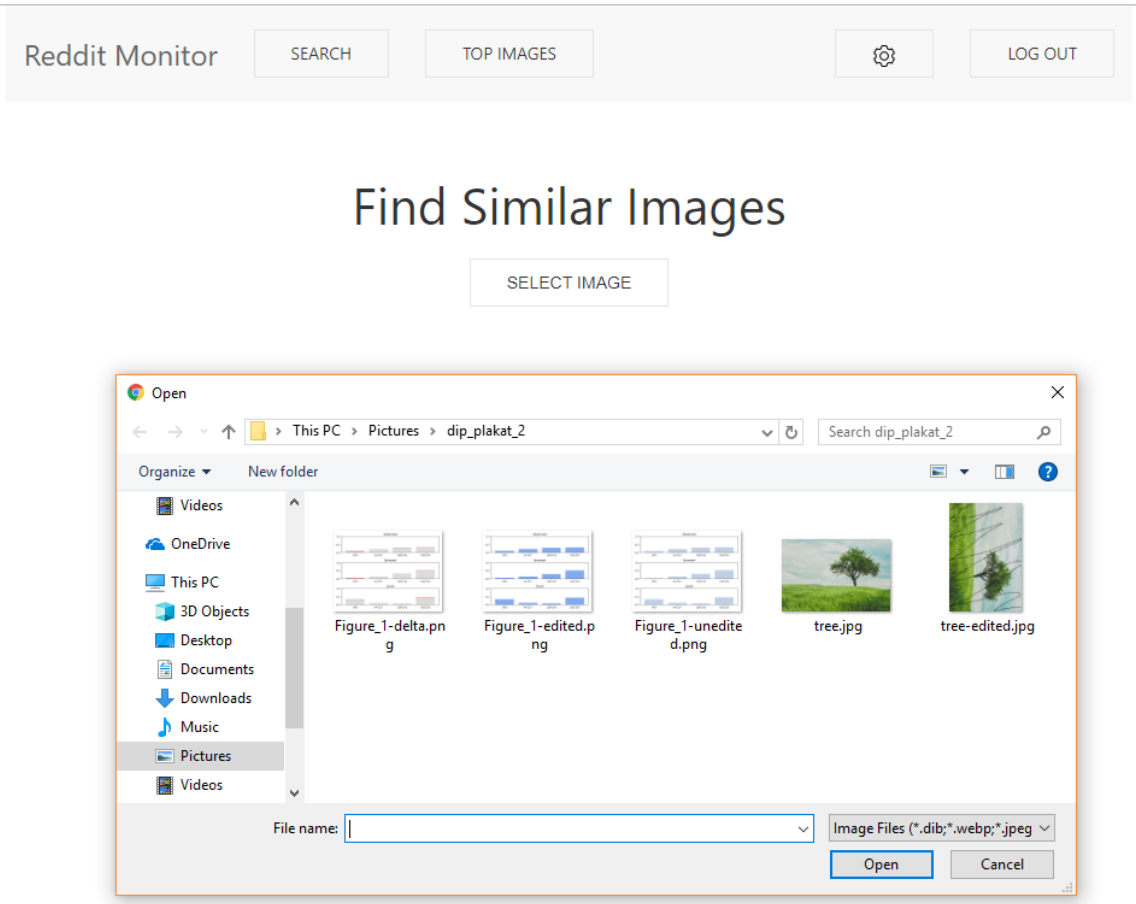


Рис. 3. Вибір зображення для пошуку

Після вибору зображення, система відобразить маску завантаження поки йде операція пошуку. Далі, у списку будуть відображені результати пошуку. При натисканні на одну з них можна побачити більш детально інформацію про зв’язані публікації.

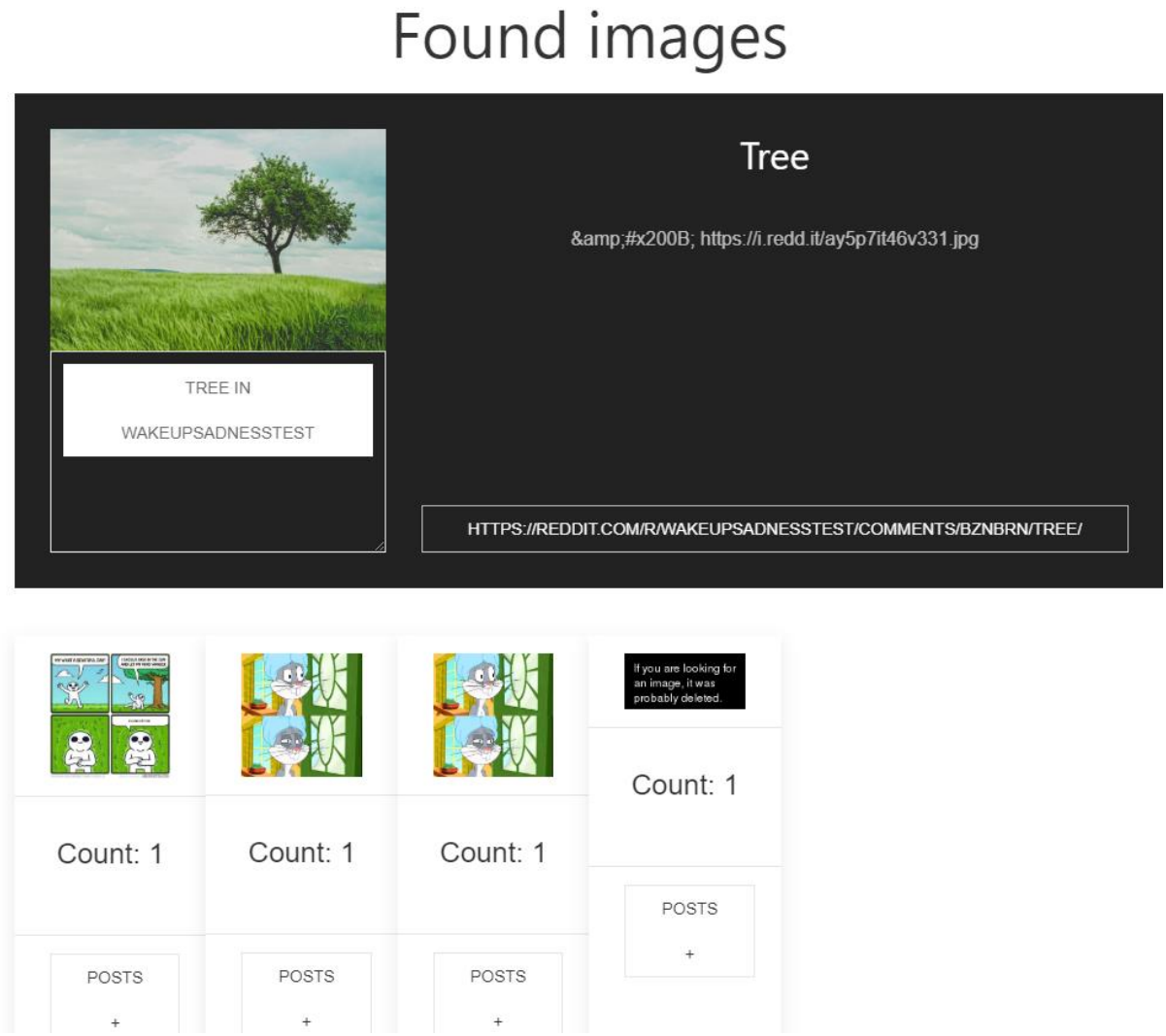


Рис. 4. Результати пошуку

5. Статистика усіх зібраних зображень

Перейшовши на сторінку усіх зображень, користувач побачить усі зображення, які зібрала система, відсортована по частоті зустрічі в різних публікаціях. Натискання на кожну з них працює так, як працює відповідна функціональна можливість на сторінці пошуку.

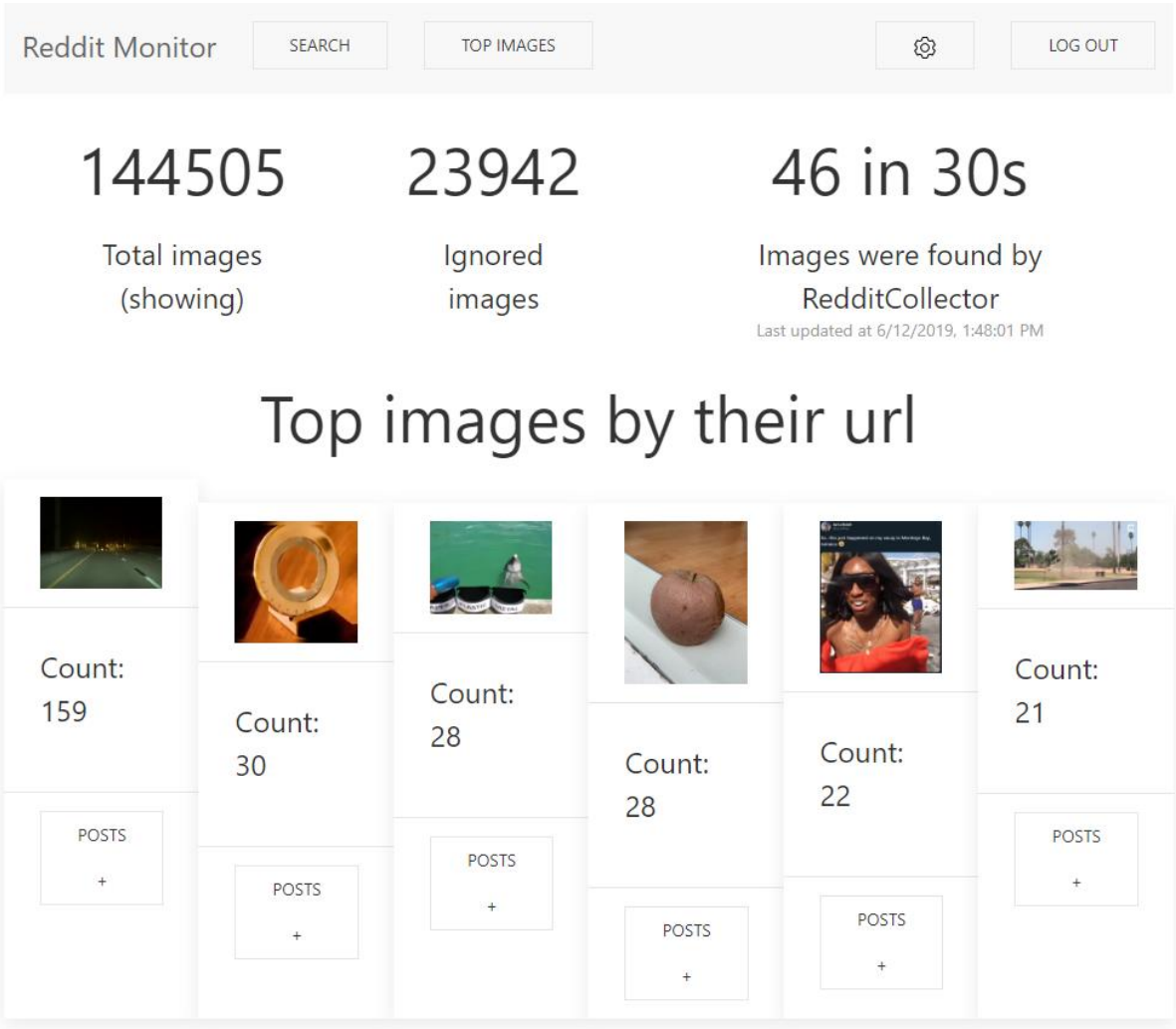


Рис. 5. Статистика усіх зображень